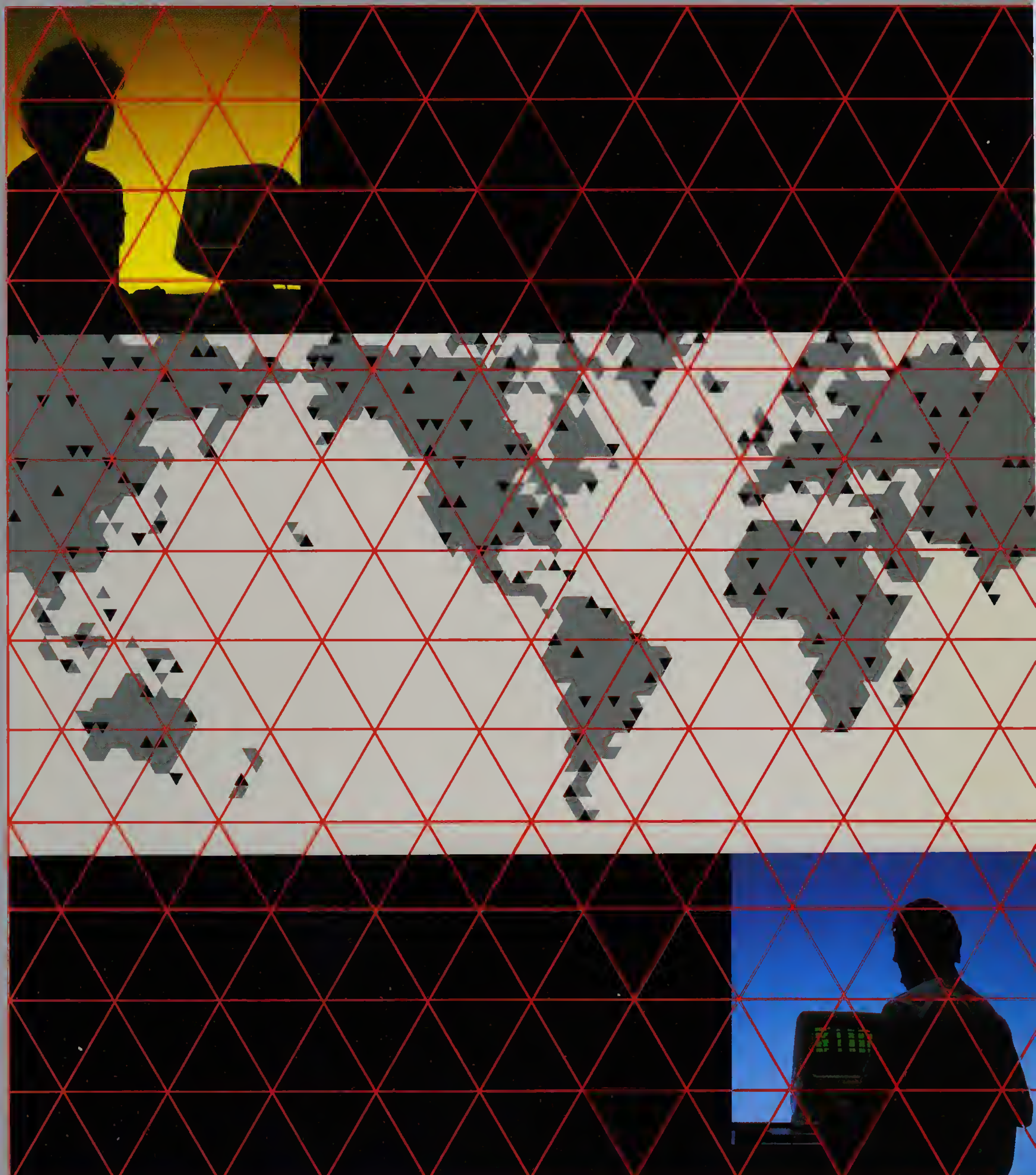
CDCNET
Network Management Entities and Layer Interfaces

Systems Programmer's Reference Manual Volume 2



60462420

CONTROL DATA

CDCNET Network Management Entities and Layer Interfaces

Systems Programmer's Reference Manual, Volume 2

Publication Number 60462420

Manual History

This manual is revision A, printed in September 1986. It documents CDCNET Software applicable to NOS and NOS/VE environments.

Contents

Figures

Tables

About This Manual

The CDCNET Systems Programmer's Reference Manual describes the CONTROL DATA® Distributed Communications Network (CDCNET) software. The CDCNET software enables you to write gateway software to support terminals, networks, and devices not currently supported by Control Data.

Audience

The Systems Programmer's Reference Manual is intended for anyone who will develop or integrate new software that must be compatible with CDCNET. This includes writing of new applications, TIPS for terminals not already supported by CDCNET, and gateways to networks using protocols foreign to CDCNET. The manual assumes the reader is familiar with CDCNET network operations.

The manual assumes the reader is familiar with and understands:

- CDCNET Network Operations
- CYBIL Programming Language
- CDCNET Systems Programmers Reference Manual, Volume 1, Base System Software
- NOS
- NOS/VE
- ISO Open Systems Interconnection (OSI) Model
- X.25 CCITT Recommendation 1980, 1984

Organization

This manual is one of a three-volume Systems Programmer's Reference Manual set that describes the CDCNET software. The following subsection contains brief descriptions of the other manuals in the set followed by a more detailed description of this manual.

Systems Programmer's Reference Manual Set

The three-volume manual set includes the following manuals.

- Volume 1, Base System Software
- Volume 2, Network Management Entities and Layer Interfaces
- Volume 3, Network Protocols

Base System Software, volume 1, describes the CDCNET DI startup and system management software: its base system software. The volume begins with an overview of each base system software component, and continues with details of the procedures and functions provided by these software components for general use.

Refer to volume 1 for additional information about procedures, intertask messages, procedures to send ITMs, BUF_PTR parameters, and BUF_PTR fields that appear frequently in the text of volume 2.

Volume 2, this volume, describes each of the network management entities (MEs) and layer interfaces for CDCNET. The volume also describes the interfaces to non-CDCNET systems. These interfaces include interfaces to NOS hosts and X.25 Packet Level networks. The information described in this volume is essential for programmers who intend to write gateway software that will reside in CDCNET.

Network Protocols, volume 3, defines CDCNET network protocols. The use of protocols enforces consistent communication between software entities and the transitions that result. The description of each protocol follows a rigid documentation guideline, called the Finite State Machine (FSM). The information described in this volume will be useful to programmers implementing Control Data Network Architecture (CDNA) on a foreign host or network.

The gateway programs you write will reside either in the DI or in a system foreign to CDCNET. The location of your gateway programs determines which volumes of the Systems Programmer's Reference Manual set will be of most help to you.

- If you are writing gateways that will be resident in the Device Interface (DI), you should understand the information in volume 1 and volume 2 of the manual set. You may want to review volume 3.
- If you are writing gateways that will be resident in a system foreign to CDCNET, you should understand the information in volume 3.

Volume 2, Network Management Entities and Layer Interfaces

Volume 2, which describes the network management entities and layer interfaces, has the following organization.

Part I introduces CDCNET.

- Chapter 1 provides an overview of the CDCNET software, especially the organization of the software.

- Chapter 2 introduces concepts important to the understanding of CDCNET software.

Part II defines and describes the external interfaces to the network MEs, those software components controlling management of the communication between networks.

- Chapter 3 describes the Initialization ME, the software that is responsible for dumping and loading DI.

- Chapter 4 describes the Routing ME, the software that creates, updates, and manages the internal tables used to route information within the network.

- Chapter 5 describes the external interfaces to the Directory ME, the software that maintains the titles and addresses which identify the individual software components in a CDCNET network.

- Chapter 6 describes the external interfaces to the File Access ME, the software that accesses and manipulates permanent files residing on host computers configured within a CDCNET network.

- Chapter 7 describes the external interfaces to the Command ME, the software that enables network operators to issue various commands to control, monitor, and maintain CDCNET networks.

- Chapter 8 describes the external interfaces to the Log ME, the software that records log messages issued by all CDCNET software components.

- Chapter 9 describes the external interfaces to the Alarm ME, the software that displays alarm messages received from all CDCNET software components.

- Chapter 10 describes the external interfaces to the Echo ME, the software that verifies a particular system in the catenet is operational by returning a message to the user in the system from which the message was received.

- Chapter 11 describes the external interfaces to the Error ME, the software that generates an Internet error report (IER) for a message in error and sends the IER to the message source. If it is not appropriate to send an error report back to the message source, the Error ME logs the error message.

- Chapter 12 describes the Clock ME, the software component that manages and synchronizes a real-time clock and calendar for every CDCNET DI.

Part III defines and describes the external interfaces to the CDCNET implementation of the OSI network layer interfaces: Session layer, Network layer, and Data Link Layer. The network layer interfaces are those software components that enable applications software, end users, terminals/workstations, and host computers to exchange information through a compatible set of protocols and interfaces.

- Chapter 13 describes the external interfaces to the Session layer. The Session layer software component enables applications such as gateway programs and TIPs to synchronize their dialogue and manage their data exchange.

- Chapter 14 describes the external interfaces to Generic Transport. The Generic Transport portion of the Transport layer delivers data to the correct destination without error and in correct sequence.

- Chapter 15 describes the external interfaces to Xerox Transport. The Xerox Transport portion of the Transport Layer software component is CDCNETs implementation of the Xerox sequence packet protocol, that performs error control and flow control for point-to-point data transmissions.

- Chapter 16 describes the external interfaces to the Internet layer. The CDCNET implementation of the Internet layer software component uses the Xerox Internet protocol to relay data between two or more systems.

- Chapter 17 describes the external interfaces to the Intranet layer. The Intranet layer software component provides the interface between specific network solutions on a particular CDCNET network.

- Chapter 18 describes the external interfaces to the mainframe channel interface (MCI) stream service routine (SSR). The MCI SSR software component enables the user to transfer data between a DI and either a NOS or NOS/VE host.

- Chapter 19 describes the external interfaces to the Ethernet Serial Channel Interface (ESCI) SSR. The ESCI SSR software component enables the user to transfer data between a DI and an Ethernet network.

- Chapter 20 describes the external interfaces to the high-speed data link channel (HDLC) SSR. The HDLC SSR software component enables the user to transfer data between DIs interconnected by either point-to-point synchronous HDLC lines or general-purpose X.25 virtual circuits.

Part IV defines and describes the user interfaces to non-CDCNET systems or networks attached to the CDCNET network. The non-CDCNET systems currently supported include those systems operating under NOS. The non-CDCNET networks currently supported include X.25 Packet Level public data networks (PDNs).

- Chapter 21 describes the external interfaces to the Block Interface Program (BIP), the software component that enables data transfer between a NOS host and a CDCNET DI.

- Chapter 22 describes the external interfaces to the Service Module (SVM), the software component that offers connection management services between a NOS host and a CDCNET DI.

- Chapter 23 describes the external interfaces to the X.25 Packet Level. This software component enables CDCNET to access X.25 Public Data Networks or Packet Data Trunks.

Appendix A provides a glossary of CDCNET terms and acronyms used in this volume.

Appendix B provides a description of the network address record.

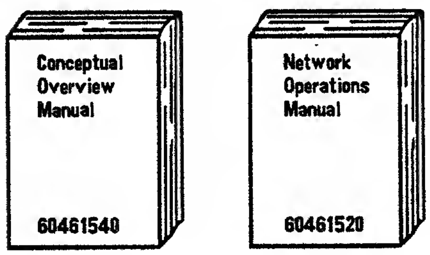
Conventions

The manual has the following conventions:

- Bold text is used to identify the fields within a record or parameters within a procedure.
- A type font that resembles computer output is used to identify CYBIL data structures and procedures within the text.
- Uppercase letters identify CYBIL types within text.
- Refer to CDCNET Volume 1, Base System Software, for additional information about the intertask message and BUF_PTR format and use.

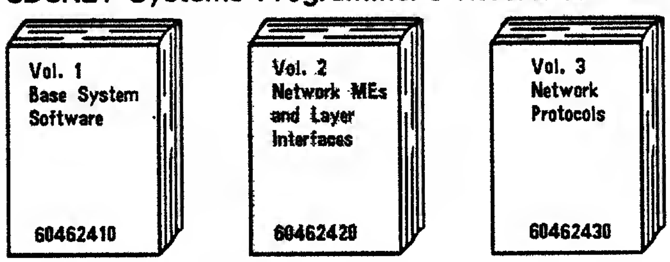
Related Manuals

Background (access as needed):



Software development manuals:

CDCNET Systems Programmer's Reference Manual



Software tools manuals:

Additional Related Manuals

Additional related manuals include the following:

Manual Title	Publication Number
CDCNET CYBIL Reference Manual	60462400
CDCNET M68000 Utilities	60462500
CDCNET MC68000 Cross-Assembler Reference Manual	60462700
CYBIL for NOS/VE System Interface	60464115

Ordering Manuals

Control Data manuals are available through Control Data Sales Offices or through:

Control Data Corporation
Literature Distribution Services
308 North Dale Street
St. Paul, Minnesota 55103

Submitting Comments

Control Data welcomes your comments about this manual. Your comments may include your opinion of the usefulness of this manual, your suggestions for specific improvements, and the reporting of any errors you have found.

You can submit your comments on the coment sheet on the last page of this manual. If the manual has no comment sheet, mail your comments on another sheet of paper to:

Control Data Corporation
Technology and Publications Division
4201 Lexington Avenue North
St. Paul, Minnesota 55126-6198

You can also submit your comments through SOLVER. SOLVER is an online facility for reporting problems. To submit a documentation comment through SOLVER, do the following:

1. Select `Report a new problem or change in existing PSR` from the main SOLVER menu.

2. Respond to the prompts for site-specific information.

3. Select `Write a comment about a manual` from the new menu.

4. Respond to the prompts.

Please indicate whether or not you would like a written response.

Overview of CDCNET Software 1

Overview of CDCNET Software 1

This chapter discusses:

- CDCNET software.
- The differences between CDCNET software running in the Network Operating System (NOS) environment and the Network Operating System/Virtual Environment (NOS/VE) environment.
- The software described in this manual and the information needed to write gateway programs.

CDCNET Software

The Control Data Distributed Communications Network (CDCNET) is a distributed data communications network that implements the Control Data Network Architecture (CDNA). CDNA is a layered architecture based on the International Standards Organization's (ISO) Open Systems Interconnection (OSI) model. The CDCNET hardware is collectively known as the device interface (DI). The software that runs on the DI is collectively known as the Distributed Communications Network Software (DCNS).

The DCNS software that resides in the DIs can be divided into the following four groups (see figure 1-1):

- Base System Software
- Network Management Entities (ME)
- Layer Software
- Interface Software

Base System Software

This group contains the software components which are responsible for establishing and maintaining an operational environment for the networking software.

This group of software is resident in every DI and is described in the CDCNET Systems Programmer's Reference Manual, Volume 1.

Network Management Entities (ME)

This group contains the software components which are responsible for control and management of the network. Some MEs have both independent and dependent components. All DIs include a complete set of dependent MEs. In a NOS environment, MDIs and MTIs contain independent MEs. In a NOS/VE environment, the independent MEs reside in the host.

Only dependent MEs offer direct interfaces to other software components. All external interfaces to dependent MEs are described in this manual. Some MEs (for example, the Routing ME, the Initialization ME, and the Clock ME) have no user interfaces, and so are not described in detail in this manual. Also, detailed information on the independent MEs, which do not have any user interfaces, will not be provided in this manual.

Layer Software

This group contains the software components that implement the CDNA layer functions. DI layer software enables application software, end users, and host computers to exchange information.

All external interfaces to CDNA layers are described in this manual.

Interface Software

Interface software refers to any set of software which help non-CDCNET systems, networks, and terminals interface to CDCNET. This group contains the software components which are responsible for interfacing NOS host systems, X.25 networks, and batch and interactive terminals to CDCNET.

Interfaces to NOS hosts and X.25 networks are described in this manual. Batch interfaces to CDCNET are described in the CDCNET Batch Device Usage manual and interactive terminal interfaces to CDCNET are described in the CDCNET Terminal Interface Usage manual.

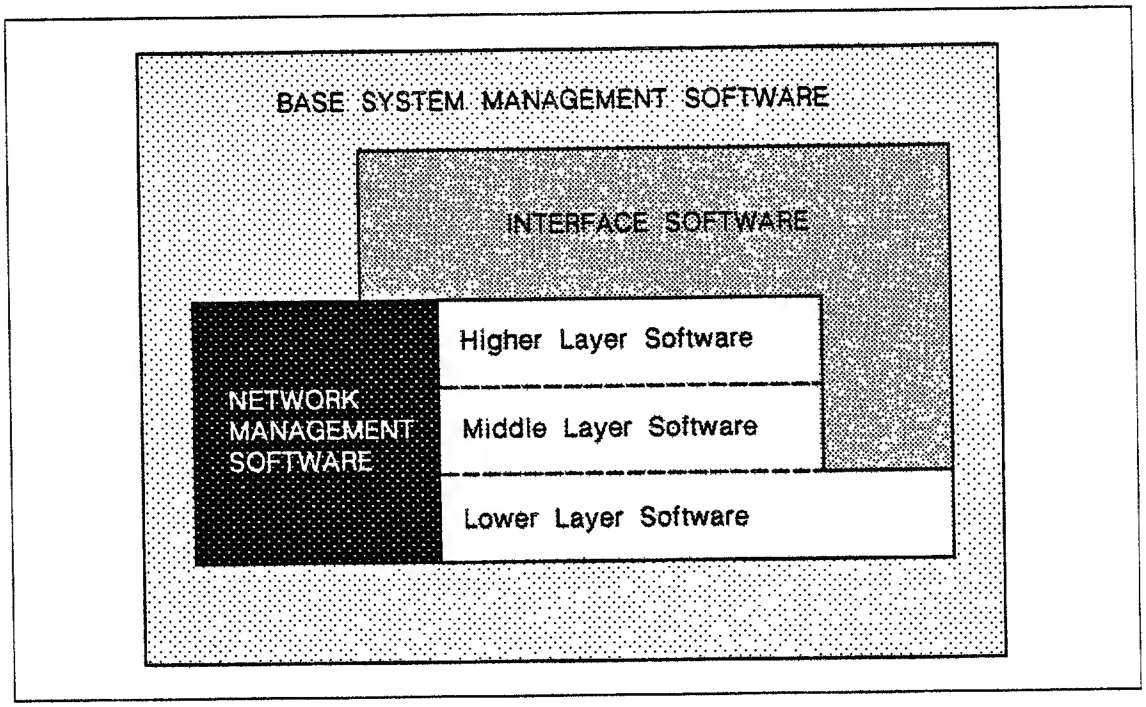


Figure 1-1. CDCNET Software

Software Described in This Manual

This manual describes the following software components:

- Layer software

  Provides communication between systems, terminals, applications, and end users connected to CDCNET

- Management Entities

  Software that helps manage a network

- Interface software

  Helps CDCNET systems connect to non-CDCNET systems.

This information is provided to enable you to develop your own gateway programs in the DI.

Software Described in Other Manuals

Besides this manual, you should read CDCNET Systems Programmer's Reference Manual, Volume 1, for information on the base system software.

For background information on protocols, read CDCNET Systems Programmer's Reference Manual, Volume 3.

CDCNET Software and CYBER Hosts

To establish a network using CDCNET, at least one CYBER host computer system must reside in a network. This host system can be either one of the following:

- CYBER 170/180 computer system running NOS
- CYBER 180 computer system running NOS/VE

Depending on which CYBER hosts operate in a network, CDCNET software either resides entirely within the DIs or is dispersed between CYBER hosts and DIs. Figure 1-2 illustrates the CDCNET software that resides in each DI variant and the CYBER host in the NOS environment. Figure 1-3 illustrates the CDCNET software that resides in each DI and the CYBER host in a NOS/VE environment.

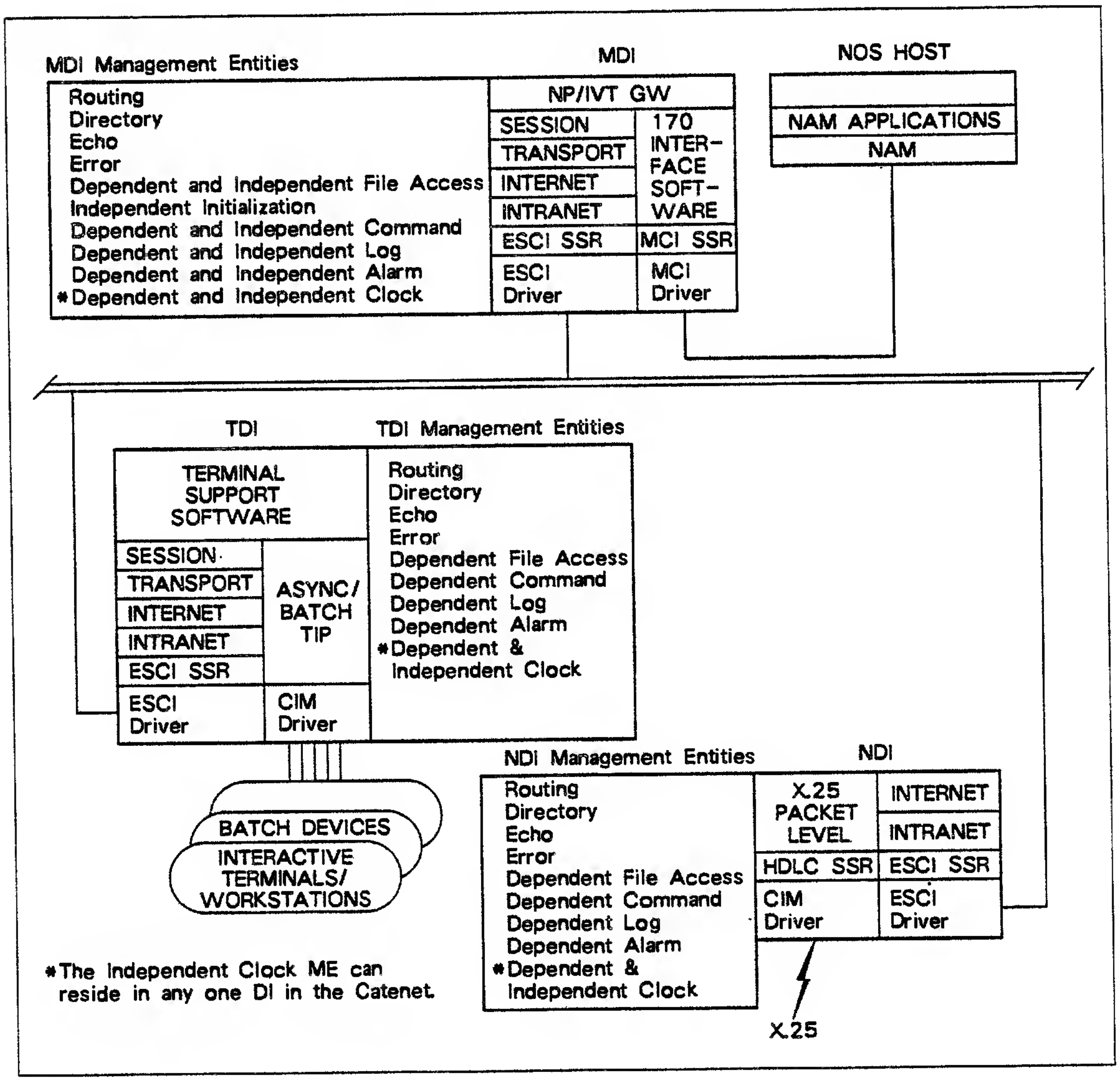


Figure 1-2. CDCNET Software in a NOS Environment

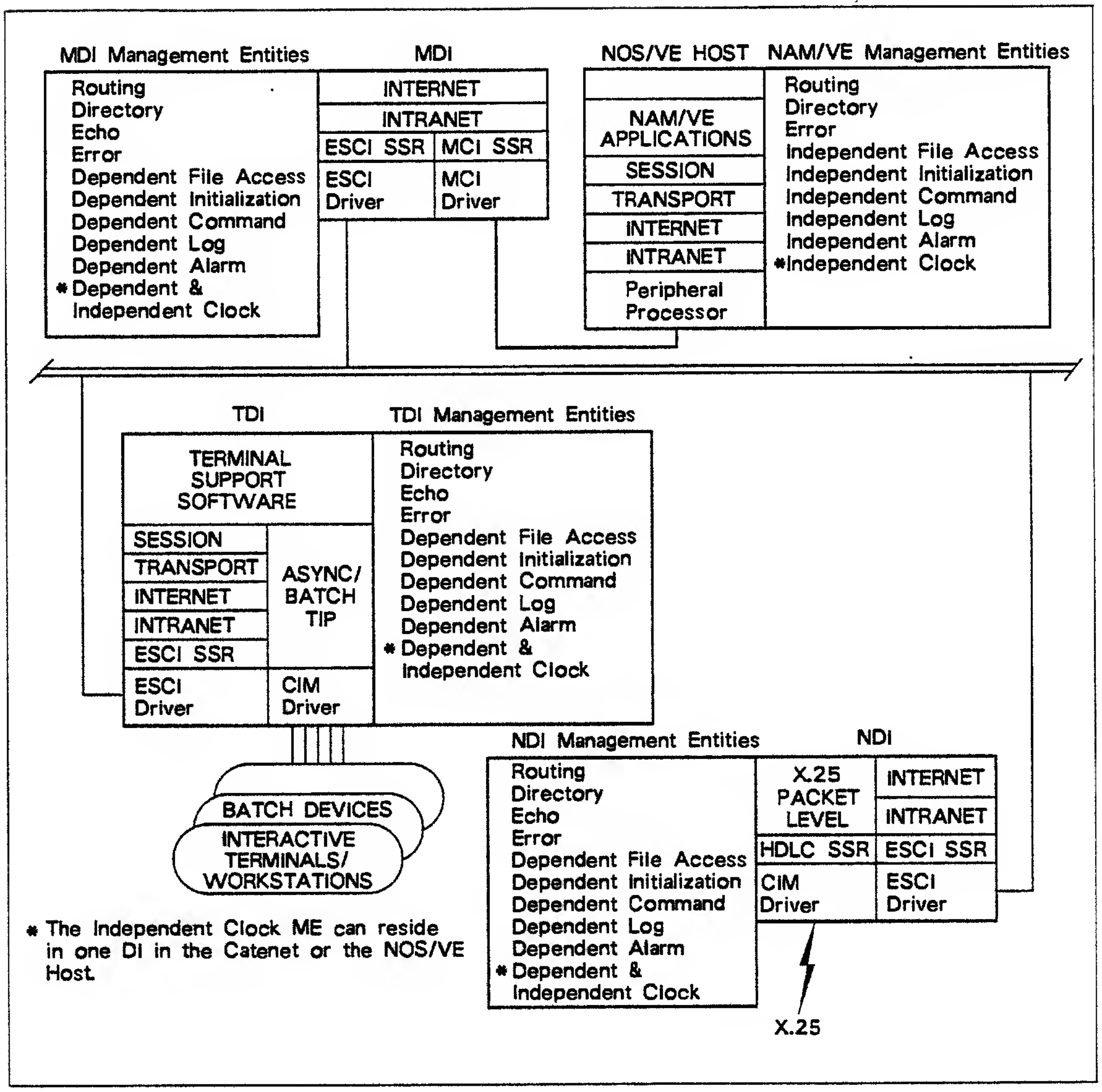


Figure 1-3. CDCNET Software in a NOS/VE Environment

Introduction to CDCNET Concepts 2

Introduction to CDCNET Concepts 2

This chapter describes the CDCNET concepts that are used throughout the manual. An understanding of these concepts will help you use the information in the manual more effectively.

Layer Principles

As shown in figure 2-1, each layer can be viewed individually as an (N) layer having an (N+1) layer as an upper boundary and an (N-1) layer as the lower boundary. The (N) layer receives services from the (N-1) layer and provides services to the (N+1) layer. The lower boundary of the physical layer is with the physical media rather than an (N-1) layer and the upper boundary of the application layer is with the application process rather than an (N+1) layer.

Each layer refers to a collection of related network procesing functions that comprise one level of a hierarchy of functions. A function or a group of functions within a layer is referred to as an entity.

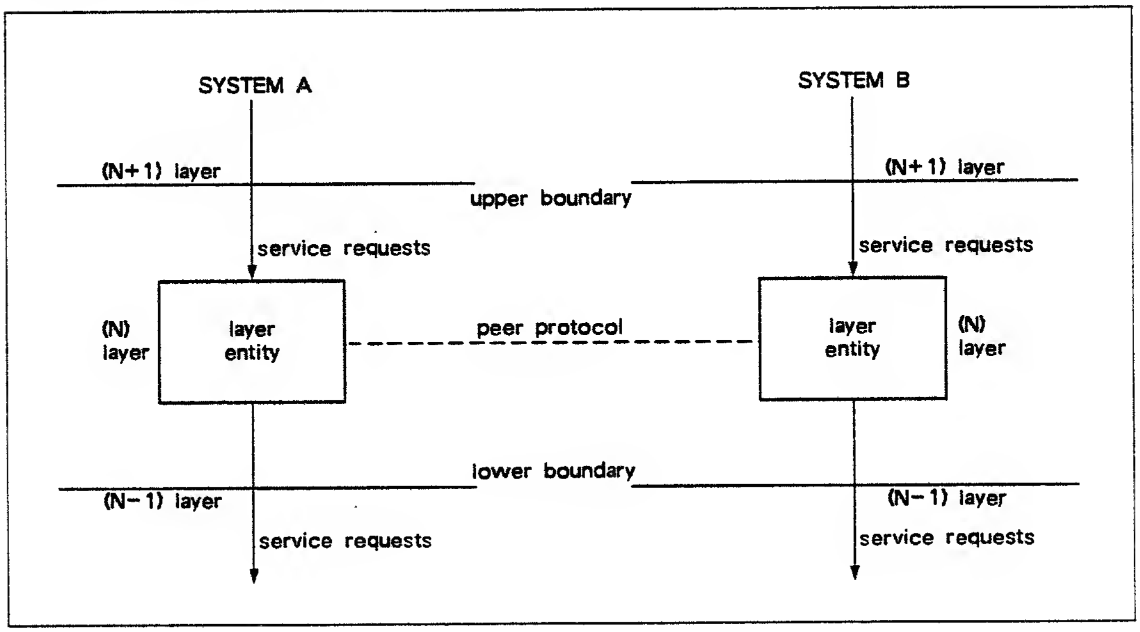


Figure 2-1. Concept of a Layer

Services

Information between layer entities is exchanged through the use of the (N-1) layer, which provides the logical connection path between (N) layer entities. In turn, each layer uses the services of the next lower layer, which are cumulatively reflected as the (N-1) services. The logical connection path represented by the dotted line in figure 2-2 is known as an (N-1) connection and provides the means for two (N) entities to communicate.

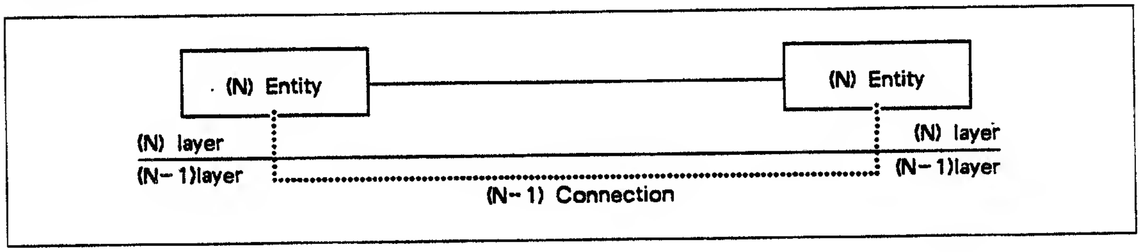


Figure 2-2. Logical Connections

Service Primitives

Interactions between entities in adjacent layers are conducted through interfaces called service primitives. See figure 2-3. The following are the different types of service primitives:

Request	This primitive allows the (N+1) layer to activate a particular service.
Indication	This primitive allows the (N) layer to indicate to the (N+1) layer that the request to activate a particular service was received from an (N) layer in another system.
Response	This primitive allows the (N+1) layer to reply to an indication service request.
Confirm	This primitive allows the (N) layer to inform the (N+1) layer that the requested service has completed.

Each of these services can have a number of associated parameters that give specific information for a particular service. For example, a connection service request can contain the destination address and the data received from the next higher layer.

In this manual, service primitives are referred to specifically by name, or collectively as interfaces.

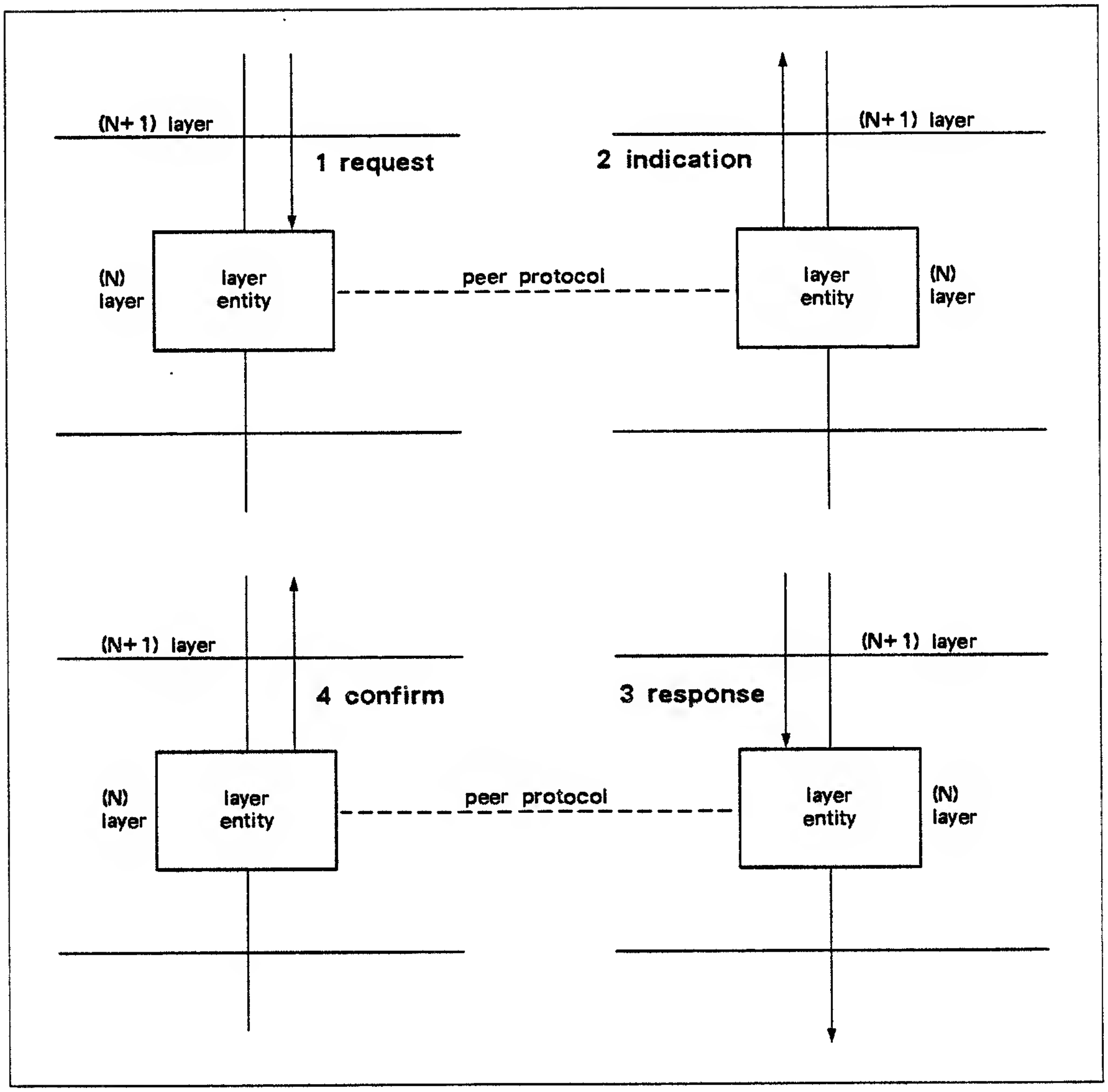


Figure 2-3. Types of Primitives

Protocols

Entities in the same layer communicate with each other using peer protocols that convey the necessary control information to support their communications. An understanding of these protocols is useful to programmers implementing Control Data Network Architecture (CDNA) on a foreign host or network. The CDCNET Systems Programmer's Reference Manual, Volume 3, gives detailed information on the protocols.

Service Access Point (SAP)

Entities of adjacent layers interconnect through layer service access points (SAPs). A SAP can be viewed as a port through which the (N) entity and (N+1) entity communicate with each other. The (N+1) entity is identified to peer (N+1) entities by the (N) SAP it uses. The other (N+1) entities know the (N) SAP by an (N) SAP identifier. More information on SAP identifiers is given under Identifiers later in this chapter.

Connections

Connections are established between peer entities. The end of a connection at a SAP is termed a connection endpoint. More than one connection endpoint can exist at a SAP. Each connection endpoint is given a connection endpoint identifier (CEPID) and is used to distinguish between different connection endpoints at a SAP. The scope of a specific CEPID is limited to a single specific SAP.

Network Solution

The term network solution is used to represent a combination of a physical medium and layers 1, 2, and 3A software which together interconnect two or more CDCNET DIs.

Catenet

A catenet is a group of connected CDCNET network solutions. This term is often used when referring to all the DIs and network solutions in a site's network.

Multicast Address

A set of systems which are connected to the same network solution can be grouped together and identified through a single address called the multicast address. CDCNET supports a special multicast address of 090025FFFFFF(16). This special address can also be considered a broadcast address, meaning it can be used to address all CDCNET systems. Multicast addresses will be supported in a future release.

Protocol Data Unit

A protocol data unit (PDU) is data that is transmitted as a unit, between peer entities.

Relay

A relay is a process where a CDCNET system receives a data unit from a locally connected network solution and transmits it to another locally connected network solution.

This concept is futher explained with the help of an illustration and an example. In figure 2-4, there are three CDCNET systems, labeled A, B, and C, and two network solutions, numbered 1 and 2. System A is connected to network solution 1, system C is connected to network solution 2 and system B is connected to both networks 1 and 2. If system A wants to send a data unit to system C, it sends it to system B on network solution 1. System B receives the data on network solution 1 and determines that it is destined for system C. System B then transmits the data unit on network solution 2 and system C receives it on network solution 2. The function performed by system B, in this example, is called a relay.

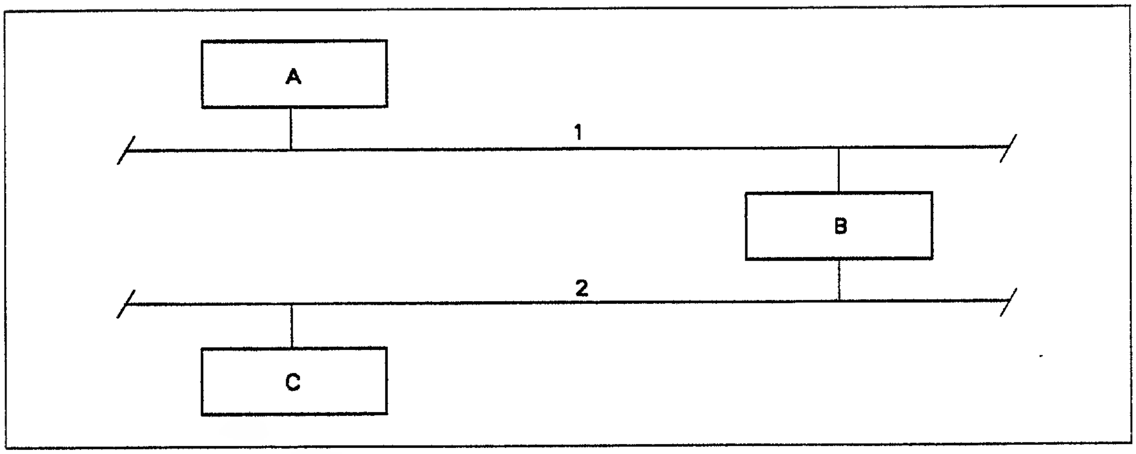


Figure 2-4. Example of Relay and Hop

Hop

The term hop is associated with the term relay. In the above example, system C is one hop away from system A. A hop count is maintained in the Internet header of each data unit and is incremented by the Internet layer each time a hop is made. If the hop counts exceed 16, the data unit is discarded. This is done to prevent the unlikely event of a data unit from wandering through the network forever. See chapter 16 in this manual for more information.

Directly Connected Network Solution

A network solution is said to be directly connected to a system if that system can directly transfer data to that network solution. In figure 2-4, network solution 1 is directly connected to systems A and B and network solution 2 is locally connected to systems C and B.

Remotely Connected Network Solution

A network solution is said to be remotely connected to a system if that system is not directly connected to that network solution, but a path to it exists. The system can then transmit data on the path to that network solution over one or more hops.

Identifiers

Identifiers, as the name implies, are used to properly identify systems, networks, and software components. The following identifiers are provided:

- Titles
- Addresses
- CEPIDs

Titles

A title is a string of 1 through 255 ASCII characters. Software components that offer services (also referred to as servers) register titles with the Directory ME to announce location and availability of a service. Clients translate titles to determine the server's availability and location.

The primary use of titles is to establish connections to the servers and services without actually knowing where they are located. Since knowledge of a physical address is not essential for establishing connections, addresses can be changed without changing a title. This feature gives the flexibility of reconfiguring servers anywhere in the catenet.

The telephone network is a good example to illustrate the relationship between titles and addresses. Just as you would use a telephone directory to find a person's telephone number, you would use the Directory ME's services to translate CDCNET titles to CDCNET addresses. See chapter 5 in this manual for more information on titles and addresses.

Addresses

CDCNET addresses are physical addresses that define the actual location of software components. All software components that offer network services are identified by the network address.

The addressing system in CDCNET is hierarchical and has some similarities to the telephone system. In the telephone system, regions in the country are assigned a number of area codes which in turn are subdivided into exchanges. Within each exchange, the subscribers are assigned a four-digit telephone number. Applying this analogy to CDCNET, the catenet represents the country and the networks within the catenet represent the area codes. Each system is the equivalent of an exchange, and finally each application in a DI corresponds to a telephone subscriber.

There are, however, some differences between the two addressing systems. Each CDCNET system address must be unique within the catenet, but the corresponding element in the telephone industry, the exchange number, does not have to be unique across the whole country. Another difference is that a CDCNET system could be physically part of more than one network, thereby acquiring more than one network address, while a telephone exchange could not be part of more than one area code.

Network Address

The network address consists of the following three components:

- Network identifier
- System identifier
- SAP identifier

Network Identifier

The network identifier specifies the network on which the system is located. Each network solution within a catenet is assigned a unique 32-bit network identifier.

The network identifier is used primarily to route data between two CDCNET networks, just as area codes are used by the telephone system to route connections.

Control Data does not manage the network identifiers, but it manages system identifiers (described in the following paragraph). A site can conveniently select its own network identifiers, keeping in mind the fact that if the site plans to connect another CDNA catenet directly without a gateway, the network identifier must be unique to ensure effective routing.

System Identifier

The system identifier specifies a particular system on a network. Each DI is assigned a unique 48-bit identification number from a pool of numbers allocated to Control Data Corporation by the Xerox Corporation. This number is written into a battery-backed RAM and is used throughout the catenet as the system identifier for that DI. Each system is assigned a system identifier. When a DI is connected to more than one physical network, the system identifier remains the same on each network; only the network identifiers are different for each network.

SAP Identifier

The SAP identifier is a 16-bit number that uniquely identifies a software component's service access point in a specified DI. The SAP identifier is the address of the Internet layer's (3B) SAP in the specified system. Some SAP identifiers are dedicated, in other words, permanently assigned to frequently used software components such as management entities. Others are nondedicated, in other words, dynamically assigned when they are needed.

Each system contains an Internet routing table to locate paths to a destination system. The network identifier-system identifier part of the network address is used to determine the route to the destination system. Once the destination system is reached, the SAP identifier is used to locate the destination 3B SAP.

Transport Address

The transport address identifies the transport SAP providing services to a particular user. CDCNET transport addresses are network addresses with the SAP identifier portion identifying the transport SAP of the transport user being addressed.

Connection Endpoint Identifier (CEPID)

See Connections, earlier in this chapter, for information on CEPIDs.

(Continued)

These MEs reside in CDCNET DIs and CDNA-compatible hosts along with other CDCNET software. The Initialization ME, Command ME, Log ME, Alarm ME, Clock ME, and File Access ME are divided into the following two parts:

- Independent functions
- Dependent functions

The Dependent MEs rely upon the Independent MEs. The management entities are divided into these two parts because it is often unnecessary and sometimes impossible to have each DI support an entire set of the required functions. For example, a subfunction of file access is concerned with direct or indirect access to some sort of mass storage. Not all CDNA systems have access to mass storage and therefore will not be able to support this subfunction.

The Dependent MEs are implemented in every DI, while the Independent MEs are implemented only in certain DIs.

The chapters in this part of the manual discuss:

- External interfaces to all Dependent MEs except Routing ME, Initialization ME, and Clock ME. Details on these three MEs are not provided because they do not have any external program interfaces.
- Brief overviews on all Independent MEs. Details on Independent MEs are not provided because they do not have external program interfaces.
- The location and functions of the MEs in NOS and NOS/VE environments.

Network Management Entities

Part II describes the network management entities (ME). This group contains the following software components, which are responsible for performing tasks related to the operation and management of the network:

- Initialization ME — Performs two major functions whenever a CDCNET DI is powered on or its reset switch is activated:
  - Broadcasts a software load request through a network.
  - The Dependent Initialization sends the reset code to the Independent Intialization ME which, in turn, uses the reset code and the information in the exception file to determine if a dump is neccessary.
- Routing ME — Creates, updates, and manages the internal tables that the Internet layer uses to route data from a source to a particular destination. It determines the most cost-effective path available between these locations.
- Directory ME — Maintains a directory of titles and associated addresses which are used to keep track of the CDCNET software components which reside in the network. This directory is maintained so that requests for particular software functions or network services can be directed to the software that can satisfy the request.
- File Access ME — Enables DIs and CDNA-compatible hosts to access and manipulate files stored on other systems residing in a CDCNET network.
- Command ME — Lets network operators monitor, control, and maintain CDCNET networks by issuing commands from either operator consoles or interactive terminals.
- Log ME — Records log messages received from CDCNET software components on a log file that resides on a host computer system.
- Alarm ME — Delivers alarm messages that are received from CDCNET software components to be displayed at the consoles/terminals maintained by network operators, or written to a host-resident file.
- Echo ME — Receives messages through CDCNET and returns them to its source; used to test if a particular DI can be reached from a particular source.
- Error ME — Alerts the Internet layer users when data transmitted through the Internet layer does not successfully reach its destination.
- Clock ME — Synchronizes the DI system clocks within the catenet.

(Continued on other side)

Initialization ME 3

Initialization ME 3

This chapter gives an overview of the Initialization Management Entity (ME). The Initialization ME does not offer interfaces that are accessible to an external user such as a gateway program. This chapter, therefore, does not contain any information on external interfaces.

Initialization ME is responsible for loading and dumping DIs. It is divided into the following two software components:

- Independent Initialization ME
- Dependent Initialization ME

Independent Initialization ME

In a NOS environment, each Mainframe Device Interface (MDI) or Mainframe Terminal Interface (MTI) that is loaded and operational contains a copy of the Independent Initialization ME which is used to load and dump other DIs. The DI that is being loaded is referred to as the remote system; the DI performing the load is referred to as the local system.

The Independent Initialization ME in the local DI uses the services of the File Access ME to read the boot file from the host. It then interfaces with the Dependent Initialization ME in a remote system to load that system.

The Independent Initialization ME uses the services of the lower layers (layers 1 through 3A) to send data to and receive data from the remote DI.

In a NOS/VE environment, the Independent Initialization ME resides on the host itself. In the initial releases, the NOS/VE host can load only directly connected systems; therefore, MDIs and MTIs are used to load and dump other DIs. The process is similar to the one just described for the NOS environment.

Dependent Initialization ME

Whenever a DI is reset, the DI issues a software load request to other DIs or hosts in its predefined network solution. This request originates from a Dependent Initialization ME. The Dependent Initialization ME resides in all CDCNET DIs and does the following:

- Broadcasts a load request to other DIs/hosts.
- Sends the reset code to the Independent Initialization ME. The Independent Initialization ME uses the reset code and the information in the exception file to determine if a dump is neccessary.

Figures 3-1 and 3-2 illustrate Initialization ME in NOS and NOS/VE environments respectively.

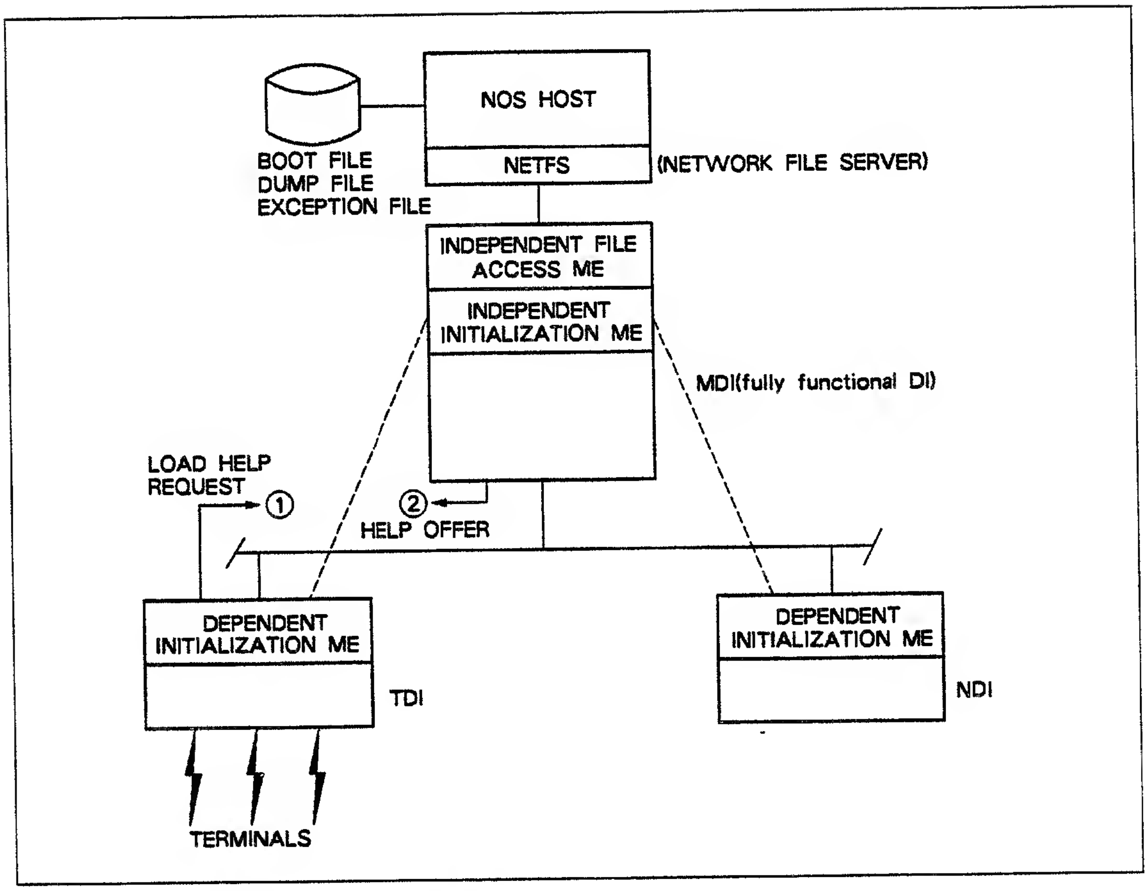


Figure 3-1. Initialization ME in a NOS Environment

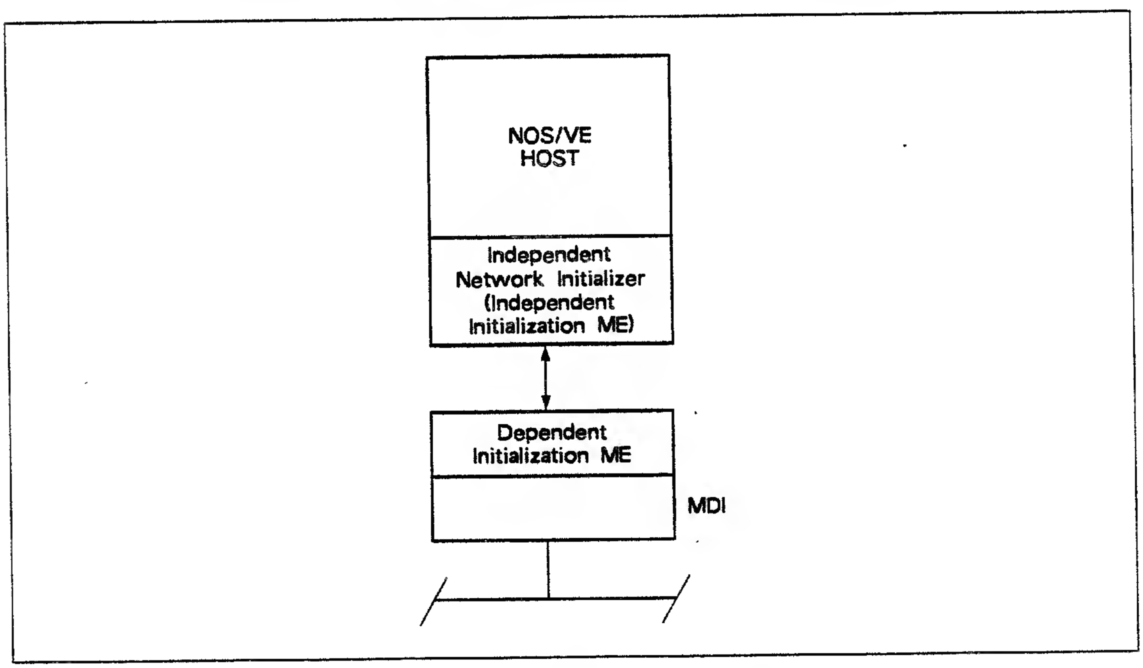


Figure 3-2. Initialization ME in a NOS/VE Environment

Initialization ME and the DI Load Process

Initialization ME plays a major role in the DI load process. The following is a step-by-step description of the process used to load CDCNET software into a remote DI. It is described here to illustrate the roles of the Independent and Dependent components of the Initialization ME.

1. The first step in loading a remote DI involves configuring the network solution connecting the local and remote DIs. The network solution is configured in the local DI through a configuration command. The command initializes the lower layer software and prepares the network solution for data transfer.

2. The Independent Initialization ME in the local DI then informs its lower layers that it is willing to receive data from any connected network solution.

   The information in the protocol header of the incoming data contains the destination address and is used by the lower layers to route data addressed to the Independent Initialization ME.

3. The Dependent Initialization ME sends a help request to the Independent Initialization ME in the local system. The Independent Initialization ME uses the information in the help request to determine if there is a need to dump the memory of the remote DI.

4. The Independent Initialization ME receives the help request and checks the exception file for any restrictions before sending a help offer. See the CDCNET Configuration and Site Administration Guide for information on the exception file.

   If the Independent Initialization ME finds there are no restrictions to loading the remote DI, it sends a help offer.

5. The Dependent Initialization ME in the remote DI receives the help offer and determines if it wants to accept the offer. If it does, it sends a help accept to the Independent Initialization ME.

6. When the Independent Initialization ME receives the help accept, it checks if a decision to dump the remote DI was made (in step 3).

7. If the remote DI memory needs to be dumped, the Independent Initialization ME uses the services of the File Access ME to create a new dump file on a network host that is accessible to it. The Independent Initialization ME then asks the remote DI to send the dumped data.

   The Independent and Dependent Initialization MEs use the initialization protocol to dump the memory of the remote DI.

   Once the dump is complete, or if the dump is not neccessary, the Initialization ME uses the services of the File Access ME to find and read the appropriate boot file on an accessible host. It then uses the services of the lower layers to transmit the information to a remote DI.

See the CDCNET Systems Programmer's Reference Manual, Volume 1 for details on system initialization and overviews of the software components that execute immediately after the DI is loaded.

In a NOS/VE environment, the Independent Initialization ME resides on the host. The host in a NOS/VE environment can be viewed as the local DI.

When the load request originates from a terminal DI (TDI) or network DI (NDI), the load request eventually will be handled by another DI which must be fully loaded and operational. This operational DI will contain an Independent Initialization ME. In turn, this Independent ME subsequently uses host-based DI configuration files to select CDCNET software that is appropriate for the requesting DI. Once the Independent Initialization ME knows which CDCNET software to select, it reads this software from host-resident files and relays the software to the original DI that requested the load. The Dependent Initialization ME within the originating DI then loads the transmitted CDCNET software into its memory.

When the load request originates from a mainframe DI (MDI), the load process is quite similar except that the load request goes directly to a CYBER host. If this host is compatible with CDNA (for example, NOS/VE systems), the host contains an Independent Initialization ME that directly communicates with the host's DI configuration files.

Routing ME 4

Routing ME 4

This chapter gives an overview of the Routing ME.

The Routing ME does not offer any interfaces that are accessible to an external user such as a gateway program. Therefore, no user procedures are described in this manual.

The Routing ME is responsible for:

- Creating, updating, and managing various internal tables that an Internet layer uses to route information. These address tables are used by the Internet layer, which is responsible for transmitting data.
- Opening and closing Internet SAPs. See chapter 16 for information on external interfaces to the Internet layer.
- Sending and receiving Routing Information Data Units.

All CDCNET DIs and CDNA-compatible hosts contain the Routing ME. Figures 4-1 and 4-2 show where the Routing ME resides in NOS and NOS/VE environments.

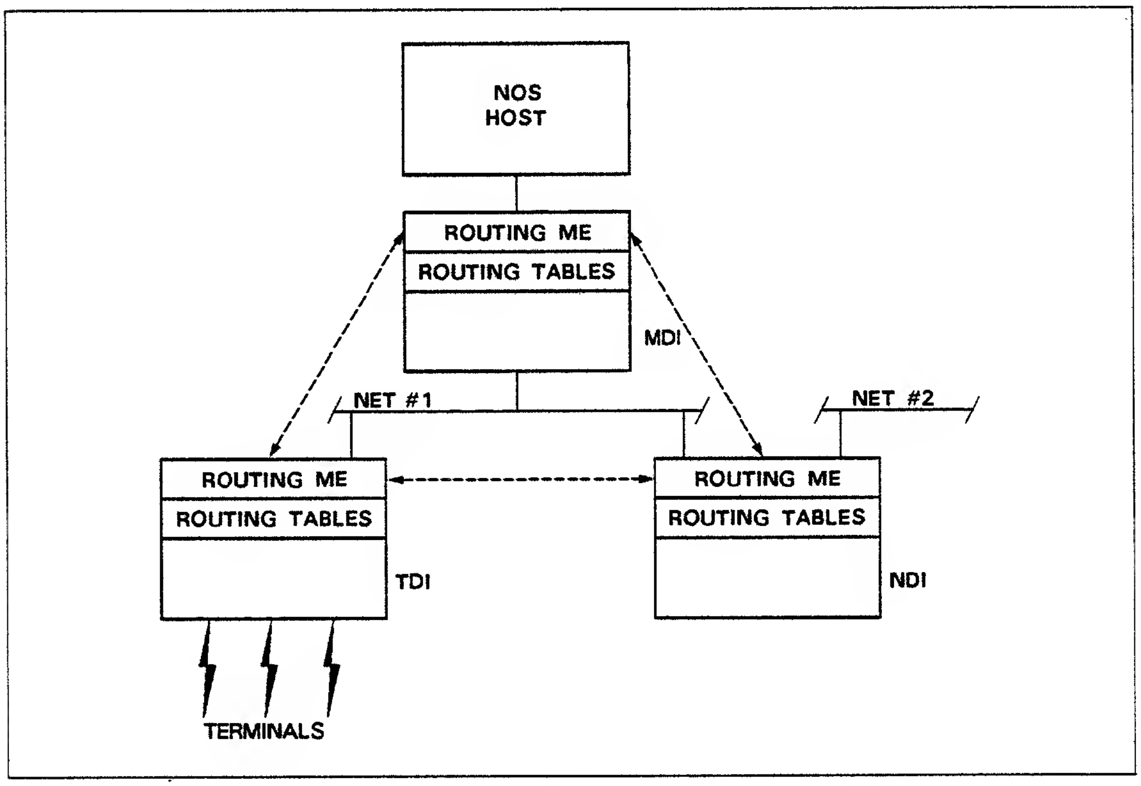


Figure 4-1. Routing ME in a NOS Environment

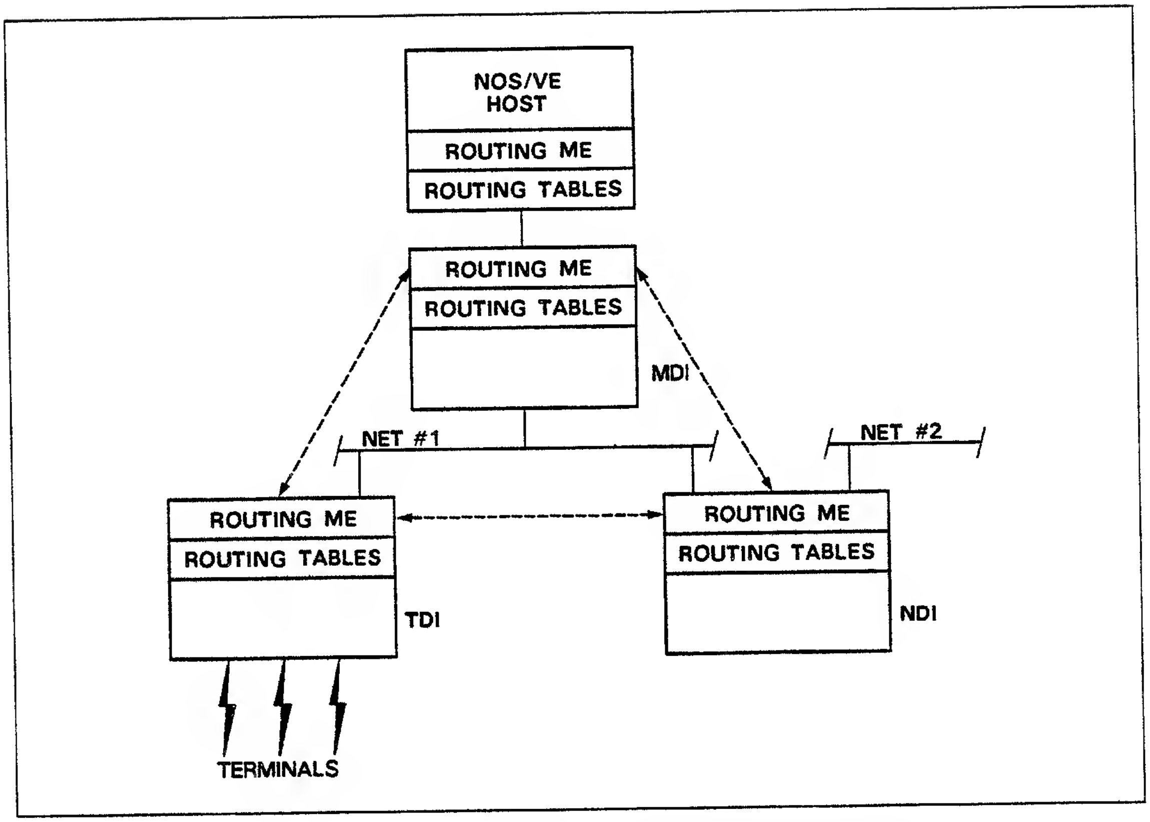


Figure 4-2. Routing ME in a NOS/VE Environment

Routing Tables

As mentioned earlier, Routing ME creates and maintains routing tables that are used by the Internet layer to route data through the network. Network routing involves finding the most efficient path in the network between the source that transmits data and the destination intended to receive it. See chapter 16 for details on the Internet layer.

All DIs and hosts that implement CDNA contain routing tables. These tables are generated when the DI or the host first comes up and contain the addresses of various DIs, hosts, and network solutions. The following are the different kinds of information provided by the various routing tables:

- Information on dedicated and dynamically assigned 3B SAPs. See chapters 2 and 16 in this manual for details on SAPs.
- Information on least cost paths.
- Information on least cost paths to accessible network solutions.
- Information on locally and remotely connected network solutions.
- Titles and addresses of local and remote networks/communities.

The routing tables maintained by any single DI or host describe only the addresses of hosts, DIs and software components that reside on the directly connected network solutions. To enable data to be exchanged outside the local area network, the Routing ME in a particular DI or host periodically (every 30 seconds) broadcasts information that identifies the specific network solution that comprises its local area network. Other Routing MEs take this information and build it into their address tables.

Directory ME 5

Directory ME 5

This chapter discusses:

- An overview of the Directory ME.
- The services required by the Directory ME.
- The services provided by Directory ME.
- Constants and common types used in the Directory ME service requests.

Overview

The Directory Management Entity (ME) resides in every device interface (DI) and NOS/VE host and is responsible for maintaining a distributed directory of titles and associated addresses. The Directory ME is provided so that a user can translate or locate the network address of a gateway or a software component. The address is required to access the services of these software components. Every software component that has services to offer registers its title and the network address at which it resides with the Directory ME.

Directory Services

The Directory ME provides two kinds of services to software components:

- Registration services
- Translation services

Registration Services

The registration services allow each entity to register its title and the corresponding address, thus announcing its availability, and the location and specific characteristics of services it offers. Registration services include the following service requests:

- Registering a title, its corresponding address, and other optional information in the directory.
- Changing information about a previously registered title.
- Deleting a title from the Directory.

Details on registration services are given in the Services Provided section of this chapter.

Translation Services

The translation services enable the user to locate and receive the address of one or more known titles. Translation services include the following service requests.

- Translating and waiting for a title.
- Translating one or more titles.
- Waiting for translation requests.
- Aborting translation requests.
- Purging translation entries.

Details on translation services are given in the Services Provided section of this chapter.

Figures 5-1 and 5-2 show Directory ME in NOS and NOS/VE environments respectively.

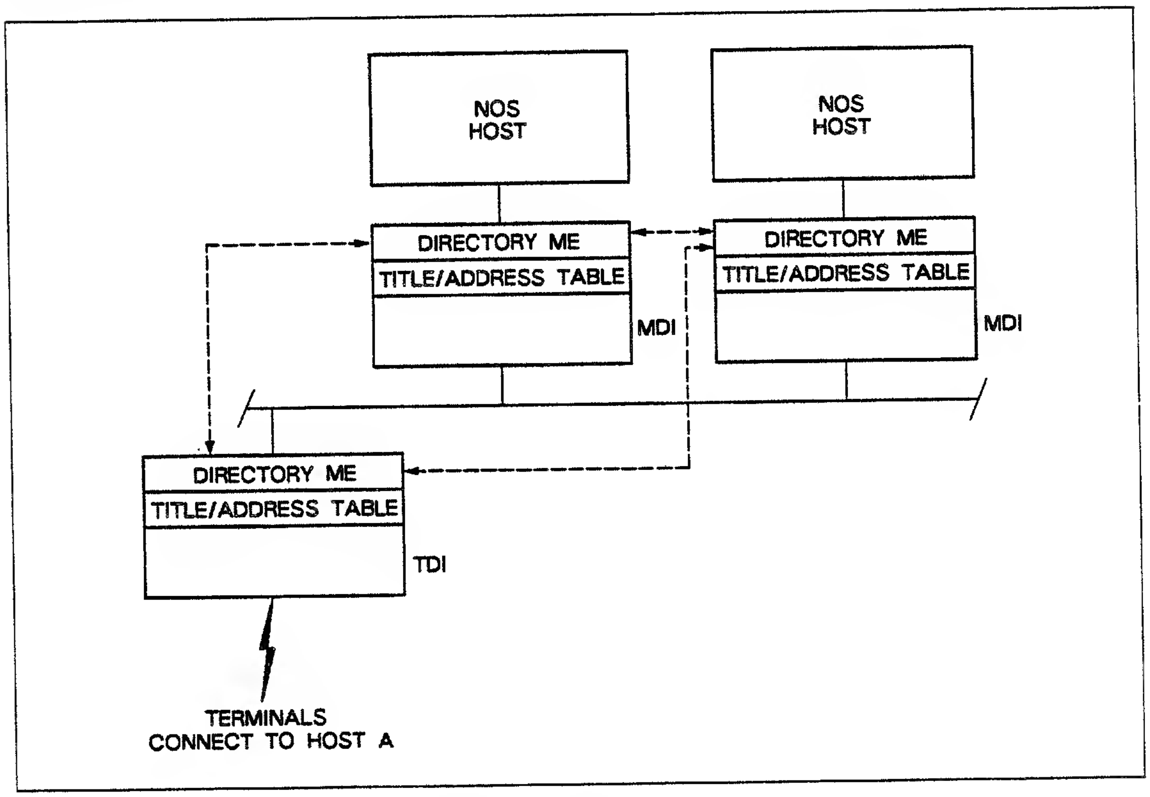


Figure 5-1. Directory ME in a NOS Environment

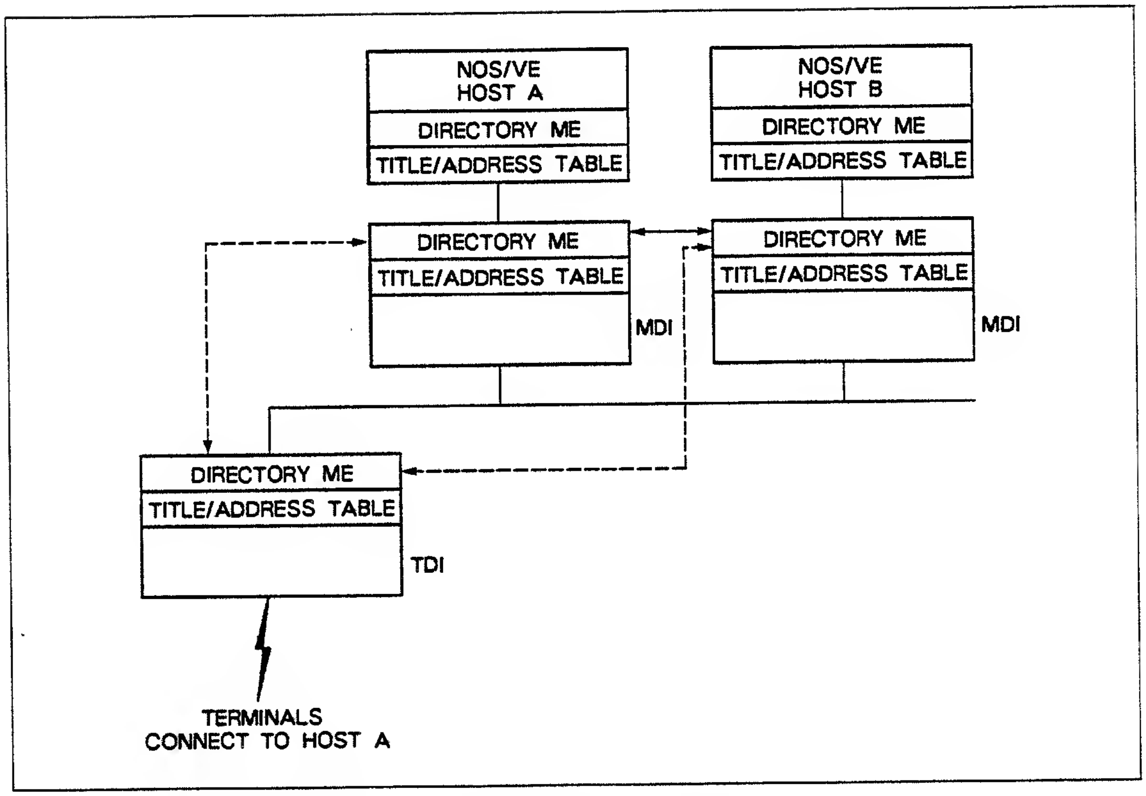


Figure 5-2. Directory ME in a NOS/VE Environment

Directory and Directory Entries

The Directory ME maintains and uses a directory to provide the registration and translation services. This directory consists of multiple directory entries. Each directory entry consists of a registered title-address pair. A title-address pair contains the following information:

- Title: any string of 1 to 255 ASCII characters (parity bit = 0). The use of a title serves two purposes:
  - It allows the user to identify the software component offering services by a logical name.
  - It allows the software component to move from one network location to another without changing the process through which the users identify it.
- Address: a field of 16 bytes that contains a network or non-network address.

  A network address consists of one of the following:
  - Network identifier and system identifier.
  - Network identifier and system identifier and 3B (Internet layer) SAP identifier.
  - Network identifier and system identifier and 3B (Internet layer) SAP identifier and Transport SAP identifier.

  A non-network address consists of a 32-bit machine-address field used by components within a system to exchange interface addresses. Non-network addresses are significant only within a local system. Therefore, directory information relative to non-network addresses is not distributed throughout the network.
- Directory Entry Identifier: Each directory entry also contains a directory entry identifier that uniquely identifies when and where that entry was created. The directory entry identifier is an 18-byte value containing the local network identifier and/or system identifier and the date and time in the binary coded decimal format.

Directory Features

The following features of Directory ME services are described here.

- Attributes
- Domain Access Control

Attributes

A directory entry may optionally contain fields called attributes, which further qualify entries. There are two kinds of attributes:

- Registration attributes
- Translation attibutes

Registration Attributes

The following is a list of registration attributes:

PRIORITY	This field indicates a priority relative to other entries with the same title.
CLASS	This field classifies the title as one of the following: Internal to CDCNET and solely for the network management use. External to CDCNET and available to all network users.
SERVICE	This optional field identifies protocols associated with the entry's address. This field specifies the set of CDNA end-to-end protocols associated with a particular directory entry. This field can be used to make sure that two users who may be communicating with each other have compatible underlying services. A title-matching criterion, specified by the user with the translation request, allows the user to receive translations from only those titles registered with the same directly accessible service.
USERINFO_PTR	The user information field provides a mechanism for extra information to be passed along to any user who requests a translation for a particular title. Up to 32 bytes of information can be passed through this parameter.
PASSWORD	This field protects an entry from unauthorized changes and deletion.
DISTRIBUTE_ TITLE	This field indicates whether or not a particular entry should be distributed gratuitously to Directory MEs in other sytems in a specified domain (see the following description of translation domain).

Translation Attributes

The following is a list of translation attributes:

RECURRENT_SEARCH	This field specifies the type of search to be performed by the Directory ME. A RECURRENT_SEARCH value set to FALSE causes the Directory ME to actively request titles from all the Directory MEs in other systems in the search domain (see following description). A RECURRENT_SEARCH value set to TRUE causes the Directory ME to wait for the titles to be distributed periodically by the other Directory MEs. With a recurrent search option, the user is automatically notified if a new server is available and the search continues until it is terminated by the user.
WILD_CARD	This field supports a set of wildcard characters that can be used to replace several strings of characters in a title. With the wildcards, the user can quickly match a number of different but related titles.

See Services Provided for more information on registration and translation attributes.

Domain Access Control

The Directory ME user can optionally exercise control over domains in which a registered title and its translation is meaningful. The following two options are available:

- Translation domain
- Search Domain

Translation Domain

This parameter allows the registration services user to identify the CDCNET systems that are to have knowledge of a particular entry. It specifies where in the network the translation request can originate and be translated. A request to translate a particular title is honored only if the request originates within the specified translation domain. For example, a title registered with a translation domain of local system cannot be obtained outside the local system. The default translation domain value is CATENET.

Search Domain

This parameter allows the translation services user to specify the domain(s) across which a search for a title is to be made. A user can specify a subset of networks and systems within the catenet to which the Directory ME should limit its title translation search. A translation of a title cannot be obtained from outside its search domain.

Domain Parameters

Special titles are used by Directory users when they want to specify a domain which may be the catenet, the individual DI, or all systems belonging to a certain community. When the Directory ME receives a request containing a domain identified by one of the following titles, it searches its own directory for that title to determine the address(es) of the system(s) in the specified domain. The following special titles are used to specify domain parameters:

Catenet

This field specifies that the domain can extend anywhere in the network and translation requests will be honored from any system in the catenet.

Local System

This field specifies that the domain is confined to the local system or an individual DI and translation requests are honored only from the local system.

Community xx

This field specifies that the domain is confined to one or more communities. Community titles are a subset of systems that are grouped together for specific purposes. xx is a community name of 1 through 31 characters. Communities are registered as the result of configuration commands. A maximum of 16 community titles can be specified. This parameter will be supported in a future release.

Directory Data Stores

Each Directory ME stores local data that contain title information useful to its local system and the catenet. Each Directory ME maintains the following data stores:

- Registration Data Store
- Translation Request Data Store
- Translation Data Store

Registration Data Store

The registration data store is maintained by all Directory MEs. It contains directory entries created by users who registered title information on the local system. Directory entries are created when the user issues a DIR_CREATE request. (DIR_CREATE is described in Services Provided, later in this chapter.)

Each directory entry in the registration data store contains the following values:

- Directory entry identifier
- Title
- Address
- Service
- Password
- User information (if supplied)
- Priority
- Translation domain
- Disribution option
- Title class

Translation Request Data Store

The translation request data store is maintained by all Directory MEs. It contains a list of all translation requests that are currently being serviced.

Each entry in the translation request data store defines one translation request. Each entry contains the following values:

- Translation request identifier
- Title (complete title or title containing wildcard character(s))
- Service (if supplied)
- Recurrent search option
- Search domain
- Title class
- Time limit
- Wildcard option

Translation Data Store

Directory MEs maintain a translation data store which contains a cache of translation data units received from other Directory MEs.

Each entry in the translation data store represents one translation data unit and contains the following values:

- The time the translation data unit was received
- Directory entry identifier
- Title
- Address
- Service
- User information (if supplied)
- Priority
- Title class
- Community titles defined for the system where the title was registered (this feature will be supported in a future release)

Services Required

The Directory ME in each DI communicates with remote Directory MEs when it receives title translation requests with search domains outside the local system. It also distributes registered titles to remote Directory MEs with a translation domain outside the local system. For the Directory ME to provide these services to its own users it, in turn, depends on the service of the following software component.

Internet Layer

To communicate with a remote Directory ME, the Directory ME in a local system uses the Internet datagram and broadcast services. The Directory ME also uses the routing module of the Internet layer that generates and broadcasts data units. It maintains tables listing network addresses and corresponding hop counts in the catenet that a Directory ME uses to communicate with a remote Directory ME. For more information, see chapter 16 in this manual.

Services Provided

This section describes the external services provided by the Directory ME to its users.

Directory ME users include the common deck DRXDIR in their calling modules. This common deck contains all the externally referenced (XREF) Directory ME procedures. The common deck DRXDIR contains calls to the common deck DRDDIR. DRDDIR defines the data structures and the parameter type declarations used by the Directory ME procedures.

The registration services of the Directory ME are described first, followed by the translation services. Under each of these services, the major data structures such as the registration control block and the translation control block, which are used in all service requests, are described first. Constants and common types used by the Directory ME and detailed explanations of returned status messages are listed at the end of the chapter.

Registration Services

As mentioned earlier, registration services allow each software component to register its title and the corresponding address, thus announcing its availability, and the location and specific characteristics of services it offers. The following is a description of the registration control block, a data structure which is used in every registration service request.

Registration Control Block

The registration control block, DIR_RCB, is the main data structure used for registration services. All registration service requests use the registration control block and all parameters that are applicable to these requests are passed within this data structure. The registration control block is a packed record defined with fields to support input parameters. Following is a description of the registration control block.

```
TYPE
  dir_rcb_rec = packed record
    title_ptr: ALIGNED ^string ( * <= max_title_len),
    community_ptr: ALIGNED ^array [1 .. * ] of string (max_community_len),
    userinfo_ptr: ALIGNED ^string ( * <= max_userinfo_len),
    password: integer,
    address: dir_address_rec,
    priority: 1 .. 0ff(16),
    service: dir_service,
    translation_domain: dir_domain,
    distribute_title: boolean,
    class: dir_class,
  recend;
```

title_ptr: ALIGNED ^string

The pointer to the title string. The title is a string of 1 through 255 ASCII (parity bit = 0) characters. MAX_TITLE_LEN is 255 characters. This is the only required field in the control block.

community_ptr: ALIGNED ^array

This field is used only if the TRANSLATION_DOMAIN field is set to LIST_OF COMMUNITIES. A maximum of 16 community titles can be specified. The community title is a string of 1 through 31 characters. This feature will be supported in a future release.

userinfo_ptr: ALIGNED ^string

This field indicates if user information was supplied. A NIL value indicates no user information was supplied. MAX_USERINFO_LEN is 32 characters.

password: integer

The password associated with a particular directory entry. The password must be supplied with a change or delete request. If this value is not supplied, the password is zero.

address: dir_address_rec

The address to be placed in the directory entry. The address is a variant record. The default address type is SYSTEM_ADDR. If the network indentifier field is zero, the network indentifier and system indentifier of the local system are stored in the address field by Directory ME for the following three address types: SYSTEM_ADDR, INTERNET_ADDR, and TRANSPORT_ADDR. For details, see Constants and Common Types, later in this chapter.

Table 5-1 shows the directory address record.

Table 5-1. Directory Address Record

Field	Content
address_type	The key field identifying the address type (type DIR_ADDRESS_TYPE). system_addr The record that contains the network indentifier, and system indentifier (type SYSTEM_ADDRESS). See appendix B for a description of this record. internet_addr The record that contains the network indentifier, the system indentifier and the 3B SAP indentifier (type INTERNET_ADDRESS). See appendix B for a description of this record. transport_addr The transport address (type GT_SAP). See chapter 14 in this manual for more information on the transport address. non_network_addr The address used by components within a system to exchange interface addresses. For example, an Ethernet device address or an X.25 DTE address which are important to gateway programs (SEQ (REP 7 of 0..0FFFF(16))). record_addr A 32-bit pointer to a cell. This field is used only when the local system is the specified domain (type ^CELL). proc_addr A 32-bit pointer to a procedure. This field is used only when the local system is the specified domain (type ^PROCEDURE).

priority: 1..Off (16)

The number which specifies the relative priority of the entry when compared with other directory entries with the same title. By convention, 1 is the highest priority. Default value is 1.

service: dir_service

The field that identifies the protocol associated with the entry's address. The following values are defined:

DIR_UNKNOWN

DIR_XEROX_INTERNET

DIR_XEROX_TRANSPORT

DIR_GENERIC_TRANSPORT

DIR_SESSION

DIR_VIRTUAL_TERMINAL

DIR_SERVICE_CDC_LAST

DIR_SERVICE_CUSTOMER_FIRST

DIR_SERVICE_CUSTOMER_LAST

Service is defined as a subrange of integers between 0 and 255. If the value is zero, this entry has a directly accessible service value of unknown. The default value is DIR_UNKNOWN. For more information, see Constants and Common Types, later in this chapter.

translation_domain: dir_domain

This field allows the user to identify the systems that are to have knowledge of its existence. A request for translation of this title is honored only if the request originates from the defined translation domain. Default translation domain value is catenet.

distribute_title: boolean

This field specifies whether or not to gratuitously distribute the directory entry throughout the translation domain. A gratuitous distribution immediately notifies a user that had requested a recurrent search that this title has been registered, or changed, or deleted. This field contains one of the following values.

TRUE	The title is to be distributed after registration.
FALSE	The title should not be distributed unless it is explicitly requested. (This is the default value.)

class: dir_class

This field identifies the originator of the title that is being registered. The values are:

CDNA_INTERNAL	Indicates the title is internal to CDCNET and is solely for the network management use. This is the default value.
CDNA_EXTERNAL	Indicates the title is external to CDCNET and is available to all network users.

Registration Control Block Initialization

An inline procedure, DIR_RCB_INIT, is provided in the common deck DRXDIR to initialize the control block to all default values.

Procedures for directory translation.

```
PROCEDURE [INLINE] dir_rcb_init
     (VAR dir_rcb: dir_rcb_rec);
     VAR rcb_templet: [XREF] dir_rcb_rec;
                 dir_rcb := rcb_templet;
PROCEND dir_rcb_init;
```

A template of the Registration Control Block is provided with the following default values:

- Title: NIL
- Service: DIR_UNKNOWN
- Translation domain: CATENET
- Distribute title: FALSE
- Community pointer: NIL
- Address type: SYSTEM_ADDR
- Address: Local SYSTEM_ADDRESS
- Password: 0
- User information pointer: NIL
- Priority: 1
- Class: CDNA_INTERNAL

Before using the registration services, the user allocates space for the registration control block. The user then initializes the registration control block using the above inline procedure. After the registration control block is initialized, the user fills in the required parameters. The title and address are the required parameters. The user can also fill in optional parameters. The registration control block is then ready to create, change, or delete directory entries.

The following example shows how a gateway program registers its title. In this example, the gateway program has successfully opened a Session layer SAP and is registering its address field, which is the service SAP identifier that was returned by the Session layer. Note that the title is external and the gateway program uses the Virtual Terminal Protocol.

```
VAR
  gw_title: string (31),
  dir_status: dir_status_codes,
  dir_id: dir_id_rec,
  dir_rcb: dir_rcb_rec,
  ivtgw_user_info: [XDCL, STATIC] string (15) := '$NP_IVT_GATEWAY';

{The following statements set up the fields in the registration control block{

  dir_rcb_init (dir_rcb);
  dir_rcb.title_ptr := ^gw_title;
  dir_rcb.address.addr_type := transport_addr;
  dir_rcb.address.transport := service_sapid;{returned by the Session layer
  dir_rcb.class := cdna_external;
  dir_rcb.service := dir_virtual_terminal;
  dir_rcb.userinfo_ptr := ^ivtgw_user_info;
  dir_create (dir_rcb, dir_id, dir_status);
  IF dir_status = dir_create_ok THEN
    Title was registered.
  ELSE
    LOG that the title was not registered.
  IFEND;
```

The following pages describe the registration service requests.

DIR_CREATE

This procedure creates a directory entry in the registration user data store.

Comdeck DRXDIR

Format DIR_CREATE (dir_rcb, dir_id, dir_status)

Input dir_rcb: dir_rcb_rec

The registration control block. See Registration Control Block, earlier in this chapter, for details.

Output dir_id: dir_id_rec

The directory entry identifier which uniquely identifies the registration request. This parameter is returned if the directory entry was successfully created.

The fields and their contents for the directory entry record (DIR_ID_REC) are shown in table 5-2:

Table 5-2. Directory Entry Record

Field	Content
system	The system address record (type SYSTEM_ADDRESS). See appendix B for details.
decclock	The date and time at which the entry was created (type BCD_TIME). See the CDCNET Systems Programmer's Reference Manual, Volume 1, for a description of BCD_TIME.

dir_status: dir_status_codes

The status indication for the processed request. The following status messages are returned for a DIR_CREATE request. For explanations, see Constants and Common Types later in this chapter.

```
dir_create_ok
dir_no_room
dir_title_err
dir_address_err
dir_userinfo_err
dir_community_err
dir_domain_err
dir_duplicate
```

DIR_CHANGE

This procedure changes a directory entry in the Registration User Data Store. The title, password, and directory entry identifier must match an existing entry in the Registration User Data Store. The user supplies a change set which specifies the parameters that are to be affected.

Comdeck **DRXDIR**

Format **DIR_CHANGE (dir_rcb, dir_change_effectors, dir_id, dir_status)**

Input

dir_rcb: dir_rcb_rec

The registration control block. See registration control block, described earlier in this chapter, for details. The title and password must be supplied to make a change.

dir_change_effectors: dir_change_set

This parameter specifies the values that are to be changed in the Directory Entry. The user information and priority fields are the only values that can be changed; all the other parameters in the registration control block remain the same. The bit for the value that has to be changed must be set by the user.

dir_id: dir_id_rec

The current identifier of the directory entry that is being changed. See table 5-2 for a description of this record.

Output

dir_status: dir_status_codes

The status indication for the processed request. The following status messages are returned for a DIR_CHANGE request. For explanations, see Contants and Common Types later in this chapter.

```
dir_change_ok
dir_no_room
dir_entry_not_found
dir_title_err
dir_userinfo_err
```

DIR_DELETE

This procedure deletes an existing directory entry in the registration user data store. The title, password, and directory entry identifier must be supplied and must match an existing registration user data store entry.

Comdeck **DRXDIR**

Format **DIR_DELETE (dir_title_ptr, dir_password, dir_id, dir_status)**

Input **dir_title_ptr: ^string (* <=max_title_len)**

The pointer to the title that is to be deleted.

dir_password: integer

The password associated with the directory entry that is being deleted.

dir_id: dir_id_rec

The directory entry identifier record. See table 5-2 for details.

Output **dir_status: dir_status_codes**

The status indication for the processed request. The following status messages are returned for a DIR_DELETE request. For explanations, see Constants and Common Types later in this chapter.

```
dir_delete_ok
dir_entry_not_found
dir_title_err
```

Translation Services

As mentioned earlier, translation services enable the user to locate and receive the address of one or more known titles. The following is a description of the translation control block, a data structure which is used in every translation service request.

Translation Control Block

The translation control block, DIR_TCB, is the main data structure used for translation services. All translation service requests use the translation control block, and all parameters that are applicable to these requests are passed within this data structure. The translation control block is a packed record defined with fields to support input parameters. Following is a description of the translation control block:

```
TYPE
  dir_tcb_rec = packed record
    title_ptr: ALIGNED ^string ( * <= max_title_len),
    community_ptr: ALIGNED ^dir_community_array
    user_id: ALIGNED ^cell,
    translation_if: ALIGNED ^procedure (dir_ticb: dir_ticb_rec;
      VAR abort_translation_request: boolean),
    time: integer,
    service: dir_service,
    search_domain: dir_domain,
    recurrent_search: boolean,
    class: dir_class,
    wild_card: boolean,
  recend;
```

title_ptr: ALIGNED ^ string (* <= max_title_len)

A pointer to the directory title that is to be translated. The title is a string (up to 255 characters). A full title can be specified or wildcard characters can be used to locate directory entries that are related in some way. For more information, see the WILD_CARD field in table 5-4.

service: dir_service

This field must match the service attribute of the registered directory entry. See DIR_SERVICE described earlier in this chapter.

community_ptr: ALIGNED ^ dir_community_array

This field is used only if the field SEARCH_DOMAIN is set to LIST_OF_COMMUNITIES. The communities are searched in the order specified in the array.

search_domain: dir_domain

This field allows the user to identify the systems where the search for the title is to be made. Default value is catenet. The Directory ME makes sure that a title registered outside the search domain will not be returned as a translation indication.

user_id: ALIGNED ^cell

An identifier supplied by the user. This value is returned by the Directory ME in the translation indication(s). The user identifies the translation request with USER_ID.

translation_if: ALIGNED ^procedure (dir_ticb: dir_ticb_rec; abort_translation_request)

The pointer to a user procedure which will receive translation indications and translation termination indications. This procedure is used only in the DIR_TRANSLATE service request, which is described later in this chapter. The Directory ME calls this procedure under the following circumstances:

- When a requested title has been found.
- When a translation request has been terminated because the search time has expired.
- When there are no more titles in the directory at a particular time.

dir_ticb: dir_ticb_rec

The translation indication control block record.

The fields and their contents for the translation indication control block record (DIR_TICB_REC) are as shown in table 5-3:

Table 5-3. Translation Indication Control Block Record

Field	Content	
user_id	The pointer that was supplied by the user with the translation request (type ^CELL).	
response_code	The key field identifying the response code (DIR_RESPONSE_CODE).	
	dir_indication	This indication informs the user that the address for a title that was requested in the translation request has been found. No duplicate indications are sent to the user unless a change was made to the title whose address is being translated.
	dir_deletion	This indication informs the user that the title has been deleted. This indication is only returned when the RECURRENT_SEARCH is set to TRUE. The following values apply to DIR_INDICATION and DIR_DELETION: title_ptr The pointer to the title string (type ALIGNED ^STRING).

(Continued)

Table 5-3. Translation Indication Control Block Record *(Continued)*

Field	Content	
		userinfo_ptr Pointer to the user information string. A NIL value indicates no user information was supplied (type ALIGNED ^STRING).
		dir_id The directory entry identifier (type DIR_ID_REC). See table 5-2 for details.
		address The address record (type DIR_ADDRESS_REC). See table 5-1 for details.
		service An ordinal that identifies the protocol associated with the directory entry. See DIR_SERVICE described earlier in the registration control block for more information (type DIR_SERVICE).
		priority The number which specifies the relative priority of the directory entry when compared with other directory entries (1..Off (16)).
	dir_indication_done	This indication is first delivered after all the translation indications in the search domain have been returned. Thereafter, this indication is sent after a translation indication for an added, changed, or deleted directory entry has been delivered.
	dir_srch_time	This indication is only returned when the RECURRENT_SEARCH field is set to FALSE. It indicates that the time has expired for this request and terminates the translation request.

abort_translation_request: boolean

The request to abort a translation request, which is set by the user.

TRUE	The translation request is to be aborted.
FALSE	The translation request should continue.

recurrent_search: boolean

This option specifies whether the Directory ME should actively request titles from the Directory MEs in the search domain or wait for the titles to be distributed by other Directory MEs. Titles requested by the RECURRENT_SEARCH option must be registered with a TRUE value set in the DISTRIBUTE TITLE field.

TRUE	The request is terminated by the user. Time is ignored.
FALSE	The request is terminated by the user or the time has expired. (This is the default.)

time: integer

The time, stated in seconds, that can be spent searching for a title. Time is not used if a recurrent search type is used. The minimum value is 0 seconds and the maximum value is 1 hour. Default value is 12 seconds. Minimum value of 0 seconds is meaningful only if the search domain is specified as the local system. Using the default time is recommended because in a future release the value will be set based on the catenet topology.

class: dir_class

This value must match the registered directory entry. A CDNA external user cannot translate CDNA internal titles. See DIR_CLASS, described earlier in the overview of this chapter.

wild_card: boolean

This field indicates if a complete title is specified or wildcards are specified in the title.

TRUE	Wildcards are used. See table 5-4 for a list of wildcard characters.
FALSE	Wildcards were not used. The title is matched character-for-character.

Table 5-4 shows the wildcard characters and their representation.

Table 5-4. Wildcard Characters

<table>
<tr><th>Wildcard</th><th>Representation</th></tr>
<tr><td>?</td><td>Represents any single character. Example:
<table>
<tr><th>Title Pattern</th><th>Titles Matched</th><th>Titles not Matched</th></tr>
<tr><td>a?a</td><td>aza</td><td>az</td></tr>
<tr><td></td><td>ala</td><td>abab</td></tr>
</table></td></tr>
<tr><td>*</td><td>Represents any string of characters of any length, including the nullstring. A maximum of five * are allowed as wildcard characters in a title. Example:
<table>
<tr><th>Title Pattern</th><th>Titles Matched</th><th>Titles not Matched</th></tr>
<tr><td>az*az</td><td>azaz</td><td>az</td></tr>
<tr><td></td><td>azlaz</td><td>azlz</td></tr>
<tr><td></td><td>azXXXaz</td><td>azXXXazYYY</td></tr>
</table></td></tr>
<tr><td>'c'</td><td>This matches the character within the quotation marks. Wildcard characters appearing in quotation marks are not interpreted as wildcard characters. Example:
<table>
<tr><th>Title Pattern</th><th>Titles Matched</th><th>Titles not Matched</th></tr>
<tr><td>a'*'*</td><td>a*22</td><td>aBC</td></tr>
<tr><td></td><td>a*Z</td><td>a'*'Z</td></tr>
<tr><td></td><td>a*</td><td></td></tr>
</table></td></tr>
<tr><td>"</td><td>This matches the quote character. If the registered title contains single quotation marks as part of the title, two single quotation marks must be used in the translation title name. Example:
<table>
<tr><th>Title Pattern</th><th>Titles Matched</th><th>Titles not Matched</th></tr>
<tr><td>c"</td><td>c'</td><td>c"</td></tr>
<tr><td></td><td>c'</td><td>c</td></tr>
</table></td></tr>
</table>

(Continued)

Table 5-4. Wildcard Characters *(Continued)*

Wildcard	Representation
[...]	This matches any single character in a group of characters or a subrange that is specified within the brackets. The group of characters is specified through one of the following:

A list	[abc]
A range	[a-z]
A combination of the above two	[a-zA-Z$#1-91]

Example:

Title Pattern	Titles Matched	Titles not Matched
a[0123]	a2	aA
	a3	A$

Translation Control Block Initialization

An inline procedure, DIR_TCB_INIT, is provided to initialize the control block to all defaults:

```
PROCEDURE [INLINE] dir_tcb_init
  (VAR dir_tcb: dir_tcb_rec);
  VAR
    tcb_templet: [XREF] dir_tcb_rec;
  dir_tcb := tcb_template;
PROCEND dir_tcb_init;
```

A template of the translation control block is provided with the following default values:

- Title: NIL
- Service: UNKNOWN
- Search domain: CATENET
- Community pointer: NIL
- User identifier: NIL
- Translation_if: NIL
- Time: DIR_DEFAULT_SEARCH_TIME

- Recurrent search: FALSE
- Class: CDNA_INTERNAL
- Wildcard: FALSE

Before using the registration services, the user allocates space for the translation control block. The user then initializes the translation control block using the above inline procedure.

After the translation control block is initialized, the user fills in the required parameters. The user can also fill in other optional parameters. The translation control block is then ready to be used in the translation service requests.

The following example shows how a terminal support program translates a gateway program's title. In this example, the title being translated is external and uses the Virtual Terminal Protocol.

```
VAR
  service_name: string (31),
  dir_tcb: dir_tcb_rec,
  dir_ttcb: dir_ttcb_rec,
  dir_status: dir_status_codes;

  dir_tcb_init (dir_tcb);
  dir_tcb.class := cdna_external;
  dir_tcb.service:= dir_virtual_terminal
  dir_tcb.title_ptr := ^service_name;
  dir_translate_and_wait (dir_tcb, dir_ttcb, dir_status);
  IF dir_status = dir_title_found THEN
      { Title found, make a call to the session call request procedure.
      { use dir_ttcb.address.transport as the DESTINATION_ADDRESS in the
      { session call request procedure.

    IFEND;

  ELSE
    Title was not found in the Directory..
    Return error.
  IFEND;
```

The following pages describe the translation service requests.

DIR_TRANSLATE_AND_WAIT

The procedure DIR_TRANSLATE_AND_WAIT is called by the users to make a single title translation request. The user is suspended until the translation is returned or the time has expired.

Comdeck DRXDIR

Format DIR_TRANSLATE_AND_WAIT (dir_tcb, dir_ttcb, dir_status)

Input **dir_tcb: dir_tcb_rec**

The directory translation request control block. See directory translation request control block, described earlier in this chapter, for details.

Output **dir_ttcb: dir_ttcb_rec**

The title translation control block.

The fields and their contents for the title translation control record (DIR_TTCB_REC) are as shown in table 5-5.

Table 5-5. Title Translation Control Record

Field	Content
dir_id	The directory entry identifier (type DIR_ID_REC). See table 5-2 for details.
address	The directory address record (type DIR_ADDRESS_REC). See table 5-1 for details.
userinfo	The user-supplied information (type STRING (MAX_USERINFO_LEN)).
priority	The current priority of this title (1..0FF (16)).
service	The directly accessible service associated with this title (type DIR_SERVICE).

dir_status: dir_status_codes

This is the status indication for the processed request. The following status messages are returned for a DIR_TRANSLATE_AND_WAIT request. For explanations, see Constants and Common Types later in this chapter.

```
dir_title_found
dir_time_expired
dir_no_room
dir_title_err
dir_community_err
```

DIR_TRANSLATE

The DIR_TRANSLATE procedure translates titles.

Comdeck **DRXDIR**

Format **DIR_TRANSLATE (dir_tcb, dir_trid, dir_status)**

Input **dir_tcb: dir_tcb_rec**

The directory translation request control block record. See the directory translation request control block, described earlier in this chapter, for details.

Output **dir_trid: dir_trid_rec**

The translation request identifier which uniquely identifies the translation request. It is returned when the translation request is confirmed.

dir_status: dir_status_codes

This is the status indication for the processed request. The following status messages are returned for a DIR_TRANSLATE request. For explanations, see Constants and Common Types later in this chapter.

```
dir_translate_ok
dir_title_err
dir_address_err
dir_domain_err
dir_community_err
```

Remarks The task issuing the DIR_TRANSLATE request is resumed immediately after the DIR_TRANSLATE call is made. This allows the task to continue processing while the title is being translated. After the processing is complete, if the task needs to be blocked, a DIR_WAIT procedure can be issued with the DIR_TRID supplied to identify the DIR_TRANSLATE call.

DIR_WAIT

The DIR_WAIT procedure allows the user to wait for a translation request to complete. The Directory ME does not return control to the user until the translation request is aborted by the user through the TRANSLATION_IF procedure or the time has expired. This assures the user that before control is returned, at least one translation has taken place or the search time has elasped.

Comdeck **DRXDIR**

Format **DIR_WAIT (dir_trid)**

Input **dir_trid: dir_trid_rec**

The translation request identifier. This parameter was returned by the DIR_TRANSLATE procedure when the translation request was confirmed.

Output None.

DIR_ABORT

The DIR_ABORT procedure aborts an outstanding directory translation request. This procedure scans the translation request data store to locate the outstanding translation request.

Comdeck DRXDIR

Format DIR_ABORT (dir_trid, dir_status)

Input dir_trid: dir_trid_rec

The translation request identifier. This parameter was returned by the DIR_TRANSLATE procedure when the translation request was made.

Output dir_status: dir_status_codes

This is the status indication for the processed request. The following status messages are returned for a DIR_ABORT request. For explanations, see Constants and Common Types later in this chapter.

```
dir_abort_ok
dir_abort_err
```

CAUTION

The DIR_ABORT procedure should not be called from within the TRANSLATION_IF procedure.

DIR_PURGE

The DIR_PURGE procedure purges an existing translation data store entry. This procedure locates the translation data entry with the same title and directory entry identifier, and deletes the entry. The user calls this procedure when a connection attempt to an address fails.

Comdeck **DRXDIR**

Format **DIR_PURGE (dir_title_ptr, dir_id, dir_status)**

Input **dir_title_ptr:**

Pointer to the title for the directory entry. This title is returned with the translation indication.

dir_id: dir_id_rec

The directory entry identifier. This parameter is returned with the translation indication. See table 5-2 for details.

Output **dir_status: dir_status_codes**

This is the status indication for the processed request. The following status messages are returned for a DIR_PURGE request. For explanations, see Constants and Common Types later in this chapter.

```
dir_purge_ok
dir_entry_not_found
```

NOTE

This procedure does not delete the registered title.

Constants and Common Types

The common deck DRRDIR defines Directory ME type declarations and the parameters. Following is a description of constants and common types required by the registration and translation services.

Constants

Constants for directory-accessible service

```
dir_unknown = 0,
dir_xerox_internet = 1,
dir_xerox_transport = 2,
dir_generic_transport = 3,
dir_session = 4,
dir_virtual_terminal = 5,
dir_service_cdc_last = 127,
dir_service_customer_first = 128,
dir_service_customer_last = 255
```

Constants for directory routines

```
max_community_len = 31,
max_community_titles = 16,
max_title_len = 255,
max_userinfo_len = 32
```

Constants for translation search times

```
dir_default_search_time = 12,
dir_min_search_time = 0,
dir_max_search_time = 3600
```

Directory return codes

```
dir_create_ok = 1,
dir_change_ok = 2,
dir_delete_ok = 3,
dir_translate_ok = 4,
dir_abort_ok = 5,
dir_purge_ok = 6,
dir_no_room = 7,
dir_entry_not_found = 8,
dir_duplicate = 9,
dir_abort_err = 10,
dir_title_err = 11,
dir_address_err = 12,
dir_userinfo_err = 13,
dir_community_err = 14,
dir_domain_err = 15,
dir_translation_if_err = 16,
dir_title_found = 17,
dir_time_expired = 18
```

dir_create_ok

The title has been added to the directory.

dir_change_ok

The title has been changed in the directory.

dir_delete_ok

The title has been deleted from the directory.

dir_translate_ok

The translation request has been confirmed.

dir_abort_ok

The translation request has been terminated.

dir_purge_ok

The translation data store entry has been purged.

dir_no_room

Currently there are no resources to allocate memory for directory entry.

dir_entry_not_found

The directory entry could not be found.

dir_duplicate

There is already an entry with same title, address, and service attribute in the directory.

dir_abort_err

The translation request could not be found.

dir_title_err

The title length is not 1 through 255 characters.

dir_address_err

The address record is invalid.

dir_userinfo_err

The length of the user information string exceeds the allowable length (MAX_USERINFO_LEN).

dir_community_err

The number of communities specified is greater than the allowed value (MAX_COMMUNITY_TITLES).

dir_domain_err

This is currently not used.

dir_translation_if_err

The address of the user procedure was not supplied for a DIR_TRANSLATE request.

dir_title_found

A title translation was returned for a DIR_TRANSLATE_AND_WAIT request.

dir_time_expired

No title was found before the time expired for a DIR_TRANSLATE_AND_WAIT request.

Common Types

```
dir_address_rec = record
  case addr_type: dir_address_type of
  = system_addr =
    system: system_address,
  = internet_addr =
    internet: internet_address,
  = transport_addr =
    transport: gt_sap,
  = non_network_addr =
    addr_data: SEQ (REP 7 of 0 .. 0ffff(16)),
  = record_addr =
    record_ptr: ^cell,
  = proc_addr =
    proc_ptr: ^procedure,
  casend
recend

dir_address_type = (system_addr, internet_addr, transport_addr,
  non_network_addr, record_addr, proc_addr)
dir_status_codes = dir_create_ok .. dir_time_expired

dir_change_set = set of (userinfo_change, priority_change)

dir_class = (cdna_internal, cdna_external)

dir_id_rec = record
  system: system_address,
  decclock: bcd_time,
recend

dir_domain = (local_system, list_of_communities, catenet)

dir_service = dir_unknown .. dir_service_customer_last

dir_status_codes = dir_create_ok .. dir_time_expired

dir_trid_rec = ^cell
```

File Access ME 6

File Access ME 6

The File Access Management Entity (ME) helps various network software components access information held in permanent files. All permanent files used by CDCNET software reside on host computers. These files serve as information databases and help the network manage functions such as loading, dumping, and configuring the DI.

File Access ME consists of the following two components:

- Independent File Access ME
- Dependent File Access ME

Independent File Access ME

The Independent File Access ME receives file access PDUs from Dependent File Access ME. The Independent File Access ME resides only in DIs with access to secondary storage. Only Mainframe Device Interfaces (MDIs) and Mainframe/Terminal Device Interfaces (MTIs) that are connected to hosts running NOS (Network Operating System) are capable of being configured with an Independent File Access ME. In a NOS/VE environment (see figure 6-2), Independent File Access ME resides on the host itself.

In a NOS environment (see figure 6-1), the Independent File Access ME connects CDCNET to a file server application on the host computer. The file server application is responsible for receiving file access requests from the Independent File Access ME and transforming them into a format acceptable to the the Network Operating System (NOS). It is also responsible for sending the processed requests back to the Independent File Access ME. All file access requests are multiplexed into one Network Products connection. This connection is initiated by the Independent File Access ME during initialization and remains active during the life of the Independent File Access ME. For more information on Network Products, see the NOS 2 Network Definition Language Reference Manual.

Since the Independent File Access ME has no external interfaces, it will not be further described in this manual.

Dependent File Access ME

The Dependent File Access ME resides in every DI in the catenet. To software components within a DI, the Dependent File Access ME appears to be providing access to local secondary storage. File access users directly interface with the Dependent File Access ME. To provide file access services, the Dependent File Access ME supports a protocol with an Independent File Access ME.

Details on Dependent File Access ME services are given in the Services Provided section.

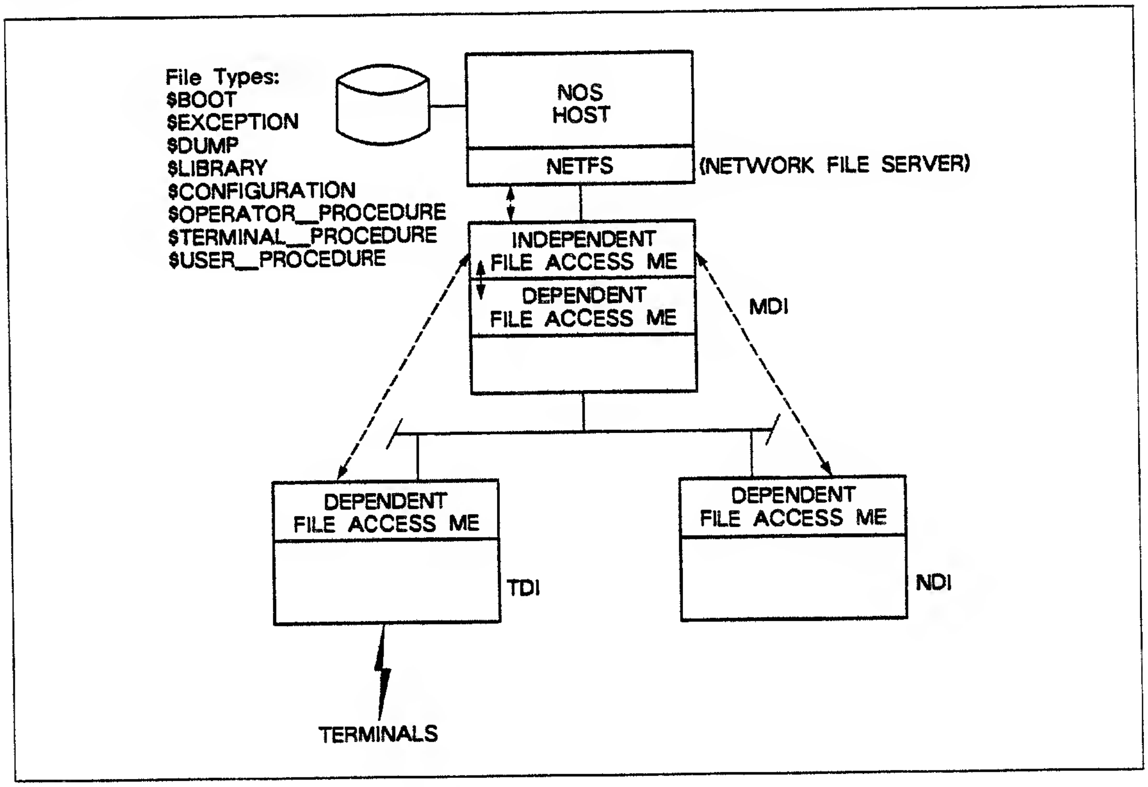


Figure 6-1. File Access ME in a NOS Environment

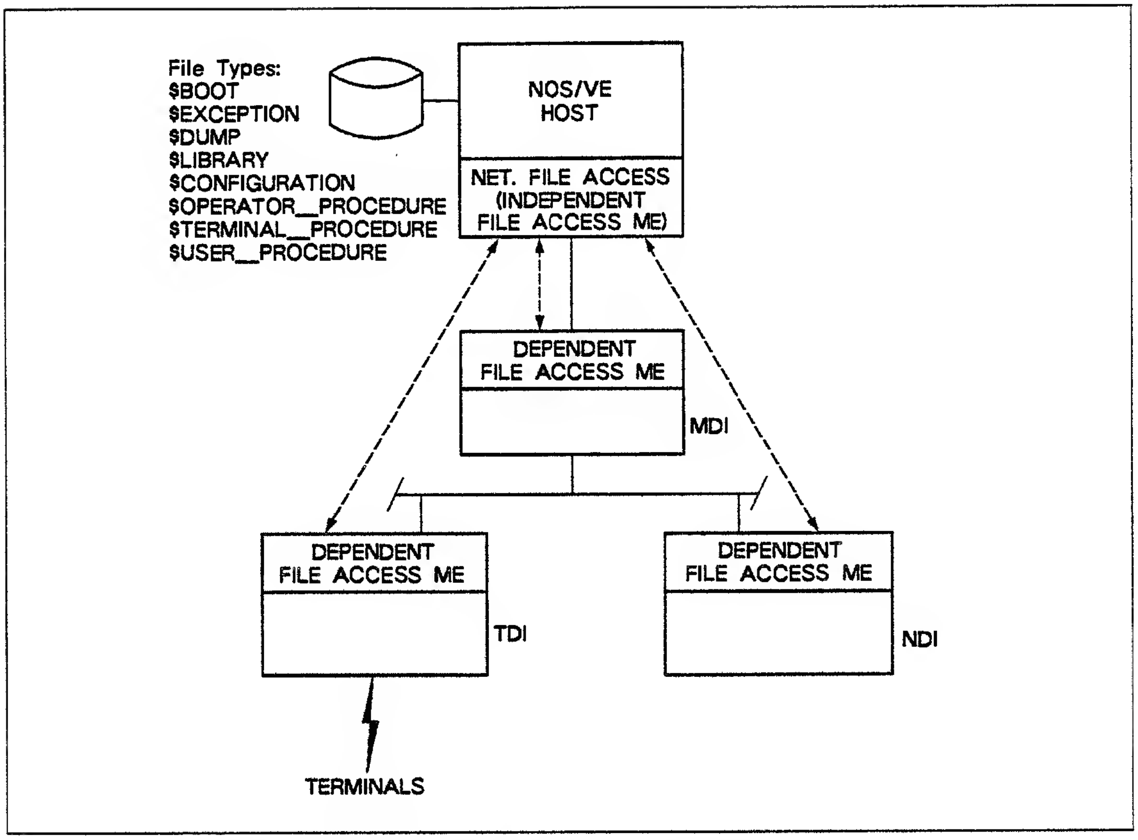


Figure 6-2. File Access ME in a NOS/VE Environment

CDCNET Files

CDCNET files are classified by the function they support. For example, configuration files support system configuration, and dump files support the dumping of a system. Each file type has a title which is maintained in the network directory along with the network address of the Independent File Access ME that supports the file type. During initialization and configuration, the Independent File Access ME registers one title for each file type it is configured to support.

The File Access ME does not restrict the number of files that can be supported or the distribution of file types across the host computers. Any host computer can be configured to support any subset of the file types that are currently defined. When an MDI is configured, it must have access to an Independent File Access ME which supports file types that are identical to the file types in its host. The following is a list of titles and file names currently supported by File Access ME:

Titles	File Names
$EXCEPTION	EXCEPTION_LIST
$BOOT	BOOT#vvvv_nnnn
$DUMP	DUMP#FULL_ssssssssssss_yymmddhhmmss DUMP#PART_ssssssssssss_yymmddhhmmss
$LIBRARY	MODULE#vvvv_m ENTRY#vvvv_e
$CONFIGURATION	CONFIGURATION#ssssssssssss
$OPERATOR_ PROCEDURE	OPERATOR_PROCEDURE#p
$TERMINAL_ PROCEDURE	TERMINAL_PROCEDURE#p
$USER_ PROCEDURE	USER_PROCEDURE#p

vvvv	A four-character hexadecimal software version number.
nnnn	The character string ESCI, HDLC, or MCI.
ssssssssssss	A 12-hexadecimal-character system identifier.
yymmddhhmmss	Date-time.
m	A 1- through 31-character module name.
e	A 1- through 31-character entry point name.
p	A 1- through 31-character procedure name.

File Access Users

File access users are software components that need access to permanent files. Following is a list of users and the file access services they use.

- The Configuration File Procurer reads configuration files.
- The Initialization ME writes dump files and reads the exception and boot files.
- The System Ancestor writes dump files.
- The Online Loader reads library files.
- The Terminal Procedure File Service module reads the terminal definition procedure and terminal user procedure files.

File Access User Interface

As shown in figure 6-3, users interface with the Dependent File Access ME through a direct call to the FILE_ACCESS procedure with a single parameter which points to a file control block. The file control block is the main data structure through which users interface to the Dependent File Access ME. All parameters that are applicable to the service requests are passed within the file control block.

The initial requests that users make are to create, open, or delete a file. For initial requests, the title of a file type and a network file name are included in the file control block. The Dependent File Access ME uses this file type and, by calling the Directory ME's translation request procedure, obtains the network address of an Independent File Access ME which supports the requested file type. The Dependent File Access ME then establishes a Generic Transport connection to the Independent File Access ME. After the initial request, the file can be accessed by a file operation service request using the same file control block.

Files can be accessed in either blocked I/O mode or nonblocked I/O mode. In blocked I/O mode, the user sends the request to the FILE_ACCESS procedure but return from this routine is blocked until the request is completed. In the nonblocked I/O mode, the user sends the request to the FILE_ACCESS procedure and specifies within the file control block the address of a procedure to be called when the request has completed.

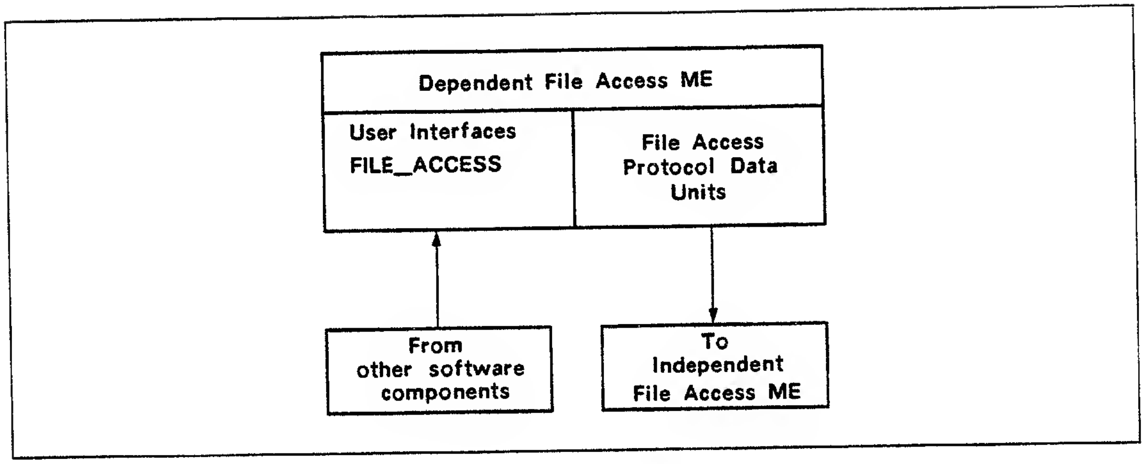


Figure 6-3. File Access ME Interfaces

Services Required

For the Dependent File Access ME to provide users access to the permanent files located in the host computer system, it depends on the services of other software components. This section briefly describes the services of each of these software components.

Generic Transport

Generic Transport transfers File Access ME PDUs between the Dependent File Access ME and the Independent File Access ME. See chapter 14 in this manual for details on Generic Transport.

Log ME

The Dependent Log ME or the Log Support Application (LSA) is used by the Dependent File Access ME to log messages that contain File Access ME protocol errors. See chapter 8 in this manual for details on Log ME.

Directory ME

Directory ME provides the Dependent File Access ME with title registration and translation services. The Dependent File Access ME requests a title translation when a file access user makes an initial request. The title translation gives the Dependent File Access ME the transport address of the Independent File Access ME that supports that particular title. The Dependent File Access ME requests Generic Transport title translation to find the address of the GT_OPEN_SAP procedure. See chapter 5 in this manual for more information on Directory ME services.

Services Provided

This section descibes the external services provided by the Dependent File Access ME to the file access user. The Dependent File Access ME provides file accessing services for a specific file by accessing the appropriate Independent File Access ME. The Independent File Access ME then accesses the requested file from the host computer's disk subsystem.

FILE_ACCESS Procedure

Users interface with the Dependent File Access ME through direct calls to the FILE_ACCESS procedure with a single parameter which points to a file control block. The file control block is the interface between the users and the Dependent File Access ME. It is a record in the user's memory and is defined with fields to support parameters for all requests and responses. File access users include the common deck CMXFAME in their calling modules. Common deck CMXFAME contains the following procedure declaration:

```
PROCEDURE [XREF] file_access (user_fcb: ^file_control);
```

The common deck CMXFAME contains calls to the common deck CMDFAME. CMDFAME contains the file control block and all the type definitions.

In the following section, the file control block is described first, followed by the various service requests and the above mentioned common procedures. In the service request descriptions, only the fields in the file control block applicable to that request are listed. The constants and common types are described at the end of the chapter.

The Dependent File Access ME provides two types of services to users in its local system:

- File specification services (later referred to as initial requests) are the initial services that the File Access ME can perform for its users. File specification services include the following service requests:
  - Creating a file (CREATE_FILE)
  - Opening a file (OPEN_FILE)
  - Deleting a file (DELETE_FILE)
- File operation services can only be used after a file specification service has been used. File operation services include the following requests:
  - Reading a file (READ_FILE)
  - Writing on a file (WRITE_FILE)
  - Repositioning within an open file (SEEK_FILE)
  - Closing a file (CLOSE_FILE)

The Dependent File Access ME also provides specific file access services for ASCII text files. Two common procedures, GET_DATA_LINE and GET_COMMAND_LINE, allow users to read a text file one line at a time. These procedures are described later in this chapter.

File Control Block

The file control block is the main data structure through which users interface to the Dependent File Access ME. All parameters that are applicable to the service requests are passed within the file control block. Users call the FILE_ACCESS procedure with the address of a file control block. Following is a CYBIL description of the file control block:

```
TYPE
file_control = record
  request_code: file_requests,
    response_procedure: ^procedure (a: ^file_control),
    fcb: ^cell,
    access_complete: boolean,
    response_code: file_responses,
    reject_code: file_reject,
    title_name: ^file_access_title,
    file_name: ^file_access_name,
    access_mode: file_access_mode,
    access_type: file_access_type,
    read_length: read_length,
    data_buffer: buf_ptr,
    origin: file_origin,
    offset: file_offset,
    user_id: ^cell,
    quality: service_quality,
    current_position: file_position,
    file_length: file_size,
    line_number: line_number,
    file_server: gt_sap,
recend;
```

request_code: file_requests

This field must contain one of the following values:

CREATE_FILE

OPEN_FILE

DELETE_FILE

READ_FILE

WRITE_FILE

SEEK_FILE

CLOSE_FILE

The user specifies one of these values in the file control block for a file access procedure call.

response_procedure: ^procedure (a: ^file_control)

The pointer to the procedure to be called when returning the file access response. If control is set to a NIL value, return is blocked until the request is complete. See blocked I/O and nonblocked I/O modes described earlier in this chapter under File Access User Interface.

fcb: ^cell

A pointer to the internal file control block which is used only by the Dependent File Access ME. This field is returned by the Dependent File Access ME as a response to an initial request. It must be included by the user in all subsequent calls to the File Access ME because it is the only link to the File Access ME's internal file control block.

CAUTION

This field is the only link to the internal file control block. Any changes made to this field may yield unpredictable results.

access_complete: boolean

This field is used in the nonblocked mode and is returned by the Dependent File Access ME to indicate access was completed. This field contains one of the following values:

TRUE Access was completed.

FALSE Access was not completed.

response_code: file_responses

This field returns a response to a service request and contains one of the following values:

REQUEST_CONFIRMED

REQUEST_REJECTED

reject_code: file_reject

An error indication from the Dependent File Access ME. It gives the reason for the unsuccessful completion of a file request. This is valid only if the RESPONSE_CODE field returns a REQUEST_REJECTED value. Error messages are listed with each service request. Explanations are given in Constants and Common Types, later in this chapter.

title_name: ^file_access_title

A required field in the file control block. The title is a string of 1 through 31 characters.

file_name: ^file_access_name

A required field in the file control block. The file name is a string of 1 through 63 characters.

access_mode: file_access_mode

This field is required for a CREATE_FILE or OPEN_FILE request and contains one of the following values:

READ_WRITE

WRITE_ONLY

READ_ONLY

access_type: file_access_type

This field is required for a CREATE_FILE or OPEN_FILE request and can be specified to be of the following types:

SEQUENTIAL

RANDOM_ACCESS

read_length: read_length

This field is required for a READ_FILE request. It specifies the number of bytes to be read from an open file.

data_buffer: buf_ptr

This field is required for a WRITE_FILE request. It is a pointer to a record called DATA_DESCRIPTOR.

origin: file_origin

This field is required for a SEEK_FILE request. It can be specified to be one of the following fields:

BEGINNING_OF_FILE

CURRENT_POSITION

END_OF_FILE

offset: file_offset

This field is required for a SEEK_FILE request. It is a number that is added to the position specified by the origin field; the resulting value is a new position within the file.

user_id: ^cell

An optional field. It is a pointer to a user-defined record.

quality: service_quality

An optional field which is reserved for future use.

current_position: file_position

This field is returned by the Dependent File Access ME and indicates the current byte position. This position, however, may not be accurate for sequential files because host record formats may not map directly to byte-addressable files.

CAUTION

This field is used by the File Access ME for internal checking. Any changes made to this field may yield unpredictable results.

---

file_length: file_size

This field indicates the length of the file in bytes. This length, however, may not be accurate for the sequential files because host record formats may not map directly to byte-addressable files.

CAUTION

Any changes made to this field may yield unpredictable results.

---

line_number: line_number

The line number that is updated by the procedures GET_COMMAND_LINE and GET_DATA_LINE. It is initialized to 0 on an OPEN_FILE or CREATE_FILE request.

file_server: gt_sap

Indicates the Independent File Access ME's Transport address.

The following example shows how the Configuration Procurer uses the services of the File Access ME to open a file.

```
    /open_configuration_file/
      WHILE (NOT file_open) AND (retry_count > 0) DO
        retry_count := retry_count - 1

        config_file.request_code := open_file;
        clp_convert_to_rjstring (mpb_ram_ptr^.system_id.upper, 16, FALSE,
                                 zero_fill, config_file_name (15, 4),
                                  convert_status);
        clp_convert_to_rjstring (mpb_ram_ptr^.system_id.lower, 16, FALSE,
                                 zero_fill, config_file_name (19, 8),
                                 convert_status);
        config_file.file_name := ^config_file_name;
        config_file.title_name := ^config_title;
        config_file.access_mode := read_only;
        config_file.line_number := 0;
        config_file.response_procedure := NIL;

        file_access (^config_file);

{ Check open status.
        IF config_file.response_code = request_confirmed THEN
          file_open := TRUE;

        ELSE
          cnfg_log_msg (cfp_file_open_error);
          wait_state := TRUE;
          delay_processing (0, open_delay, 0, 0);
          wait_state := FALSE;

{ If the operator has entered a BYPASS_CONFIGURATION command, do not try to
      get the configuration file again.

           IF bypass_config_flag THEN
            cnfg_log_msg (cfp_bypass_config);
            config_status := config_ok;
            RETURN;

          IFEND;

        IFEND; { Check open status.

      WHILEND /open_configuration_file/;
```

The following pages describe the file access service requests.

CREATE_FILE

This request creates a file. A request code of CREATE_FILE in the file control block for a file access procedure call initiates this request. If a file of the same name exists and can be modified by the user, it is truncated to zero length. If there is no duplicate file, a new file of zero length is created.

Input The user sets up the following fields as input in the file control block:

request_code: file_requests

This field contains the following value:

CREATE_FILE

response_procedure: ^procedure

title_name: ^file_access_title

file_name: ^file_access_name

access_mode: file_access_mode

access_type: file_access_type

Output The Dependent File Access ME returns the following fields in the file control block:

fcb: ^cell

access_complete: boolean

response_code: file_responses

reject_code: file_reject

file_length: file_size

file_server: gt_sap

Following is a list of reject codes applicable to this request. For explanations, see Constants and Common Types later in this chapter.

```
unspecified_error
security_error
insufficient_space
i_o_error
bad_file_name
```

OPEN_FILE

This request opens a file that has already been created. A request code of OPEN_FILE in the file control block for a file access procedure call initiates this request. The user needs to open a file before accessing it. The file opens if a file of the specified name exists and the requested access mode is allowed.

Input The user sets up the following fields as input in the file control block.

request_code: file_requests

This field contains the following value:

OPEN_FILE

response_code: file_responses

title_name: ^file_access_title

file_name: ^file_access_name

access_mode: file_access_mode

access_type: file_access_type

Output The Dependent File Access ME returns the following fields in the file control block:

fcb: ^cell

access_complete: boolean

response_code: file_responses

reject_code: file_reject

file_length: file_size

file_server: gt_sap

Following is a list of reject codes applicable to this request. For explanations, see Constants and Common Types later in this chapter.

```
unspecified_error
security_error
i_o_error
file_does_not_exist
bad_file_name
```

DELETE_FILE

This request is used to delete a file that is no longer required. A request code of DELETE_FILE in the file control block for a file access procedure call initiates this request.

Input The user sets up the following fields as input in the file control block.

request_code: file_requests

This field contains the following value:

DELETE_FILE

response_procedure: ^procedure

title_name: ^file_access_type

file_name: ^file_access_name

Output The Dependent File Access ME returns the following fields in the file control block:

access_complete: boolean

response_code: file_responses

reject_code: file_reject

Following is a list of reject codes applicable to this request. For explanations, see Constants and Common Types later in this chapter.

```
unspecified_error
security_error
i_o_error
file_does_not_exist
bad_file_name
```

READ_FILE

This request is used to read data from an open file. A request code of READ_FILE in the file control block for a file access procedure call initiates this request.

Input The user sets up the following fields as input in the file control block.

request_code: file_requests

This field contains the following value:

READ_FILE

response_procedure: ^procedure

read_length: read_length

Output The Dependent File Access ME returns the following fields in the file control block:

access_complete: boolean

response_code: file_responses

reject_code: file_reject

data_buffer: buf_ptr

current_position: file_position

Following is a list of reject codes applicable to this request. For explanations, see the Constants and Common Types section later in this chapter.

```
unspecified_error
i_o_error
beyond_end_of_file
fcb_active
unknown_fcb
bad_byte_count
security_error
```

WRITE_FILE

This request is used to write data to an open file. A request code of WRITE_FILE in the file control block for a file access procedure call initiates this request.

Input The user sets up the following fields as input in the file control block.

request_code: file_requests

This field contains the following value:

WRITE_FILE

response_procedure: ^procedure

data_buffer: buf_ptr

Output The Dependent File Access ME returns the following fields in the file control block:

access_complete: boolean

response_code: file_responses

reject_code: file_reject

current_position: file_position

Following is a list of reject codes applicable to this request. For explanations, see Constants and Common Types later in this chapter.

```
unspecified_error
i_o_error
insufficient_space
unknown_fcb
fcb_active
security_error
bad_byte_count
```

SEEK_FILE

This request is used to change the current position within an open file. It is allowed only on random files. A request code of SEEK_FILE in the file control block for a file access procedure call initiates this request.

Input The user sets up the following fields as input in the file control block.

request_code: file_requests

This field contains the following value:

SEEK_FILE

response_procedure: ^procedure

origin: file_origin

offset: file_offset

Output The Dependent File Access ME returns the following fields in the file control block:

access_complete: boolean

response_code: file_responses

reject_code: file_reject

Following is a list of reject codes applicable to this request. For explanations, see Constants and Common Types later in this chapter.

```
no_seek_on sequential_file
fcb_active
unknown_fcb
invalid_file_position
illformatted_request
```

CLOSE_FILE

This request is used to close a previously opened or created file. A request code of CLOSE_FILE in the file control block for a file access procedure call initiates this request.

Input The user sets up the following fields as input in the file control block.

request_code: file_requests

This field contains the following value:

CLOSE_FILE

response_procedure: ^procedure

Output The Dependent File Access ME returns the following fields in the file control block:

access_complete: boolean

response_code: file_responses

reject_code: file_reject

Following is a list of reject codes applicable to this request. For explanations, see Constants and Common Types later in this chapter.

```
unspecified_error
i_o_error
fcb_active
unknown_fcb
```

GET_DATA_LINE

The GET_DATA_LINE procedure allows the user to read a text file line by line. The data lines are passed to the caller one at a time with trailing spaces removed. A data line is a string of ASCII characters terminated by the unit separator [1F(16)]. The unit separator or end of file terminates a line regardless of where it was encountered. To use this procedure, a user first initializes the user file control block (USER_FCB) by calling the file access procedure with an OPEN_FILE request.

Comdeck CMXGDL

Format **GET_DATA_LINE (user_fcb, line, read_status)**

Input **user_fcb: ^file_control**

This parameter is a pointer to the file control block. For explanations, see File Control Block earlier in this chapter.

Output **line: ost$string**

This type is described in chapter 7 of this manual.

read_status: read_file_status

This is the status indication of the processed request. The following status messages are returned for this procedure. For explanations, see Constants and Common Types later in this chapter.

```
read_ok
read_eof
line_too_long
access_error
```

GET_COMMAND_LINE

The GET_COMMAND_LINE procedure allows the user to read a file of System Command Language (SCL) commands, one command at a time. Physical lines are read until an entire command line is put together.

Format GET_COMMAND_LINE (user_fcb, command, read_status)

Comdeck CMXGCL

Input **user_fcb: ^file_control**

This parameter is a pointer to the file control block. For explanations, see File Control Block, earlier in this chapter.

Output **command: ost$string**

This type is described in chapter 7 of this manual.

read_status: read_file_status

This is the status indication of the processed request. The following status messages are returned for this procedure. For explanations, see Constants and Common Types later in this chapter.

```
read_ok
read_eof
line_too_long
access_error
```

Remarks Before this procedure passes a command line back to the user, it makes the following changes (if necessary):

- Compresses multiple blanks down to a single blank if they are not within a string.
- Replaces comments with a single blank if the comments are not within a string.
- Removes leading and trailing blanks and blank lines.
- Processes ellipsis (..) at the end of a line. It eliminates the ellipsis and appends the next line.
- Maintains a running line number counter. (The counter is set to zero by Dependent File Access ME when the file is opened.)

Because of this editing, the line number returned refers to the first physical line read as part of the command, excluding the blank/comment lines. A physical line is ASCII characters terminated by the unit separator [1F(16)]. To use this procedure, a user initializes the user file control block (USER_FCB) by calling the FILE_ACCESS procedure with an OPEN_FILE request.

The following example shows how a software component, the Configuration Procurer, uses the GET_COMMAND_LINE procedure. For information on the Configuration Procurer, see the CDCNET Systems Programmer's Reference Manual, Volume 1.

```
    config_file_ptr := ^config_file;
    command_file_status := read_ok;

  /process_configuration_commands/
    WHILE command_file_status <> read_eof DO
      get_command_line (config_file_ptr,
        command, command_file_status);
      command_line_number := config_file_ptr^.line_number;

{ Check command file status.
      IF command_file_status = read_ok THEN
        clp_process_command (command, command_status);

        IF command_status.normal THEN
          cnfg_log_msg (cfp_normal_cmd_response);

        ELSE
          cnfg_log_msg (cfp_abnormal_cmd_response);
        IFEND;

      ELSEIF command_file_status = line_too_long THEN
        cnfg_log_msg (cfp_ill_formed_text_line);

      ELSEIF command_file_status = access_error THEN
        reset_di (configuration_file_read_error);

      ELSE
        ; { end of file.

      IFEND; { Check command file status.

      WHILEND;
```

Constants and Common Types

The common deck CMDFAME defines File Access ME type declarations and parameters. Following is a description of constants and common types required by the file access service procedures

Constants

```
boot_title = '$BOOT'
cnfg_title = '$CONFIGURATION'
dump_title = '$DUMP'
except_title ='$EXCEPTION'
lib_title =   '$LIBRARY'
op_proc_title='$OPERATOR_PROCEDURE'
term_proc_title='$TERMINAL_PROCEDURE'
user_proc_title ='$USER_PROCEDURE'

max_byte_file_size = 07fffffff(16)
max_file_name_len = 63
max_title_name_len = 31
```

Common Types

```
file_access_mode = (read_write, write_only, read_only)

file_access_name = string ( * <= max_file_name_len)

file_access_title = string ( * <= max_title_name_len)

file_access_type = (sequential, random)

file_offset = integer

file_origin = (beginning_of_file, current_position, end_of_file)

file_position = 0 .. max_byte_file_size

file_reject = (
unspecified_error,
security_error,
insufficient_space,
i_o_error,
file_does_not_exist,
invalid_file_position,
file_service_unavailable,
protocol_error,
unexpected_file_close,
no_seek_on_sequential_file,
bad_byte_count,
bad_file_name,
beyond_end_of_file,
fcb_active,
illformatted_request,
purge_busy,
unknown_fcb,
usage_conflict,
read_ok,
read_eof,
line_too_long,
access_error
```

unspecified_error	A protocol error has occurred between an Independent File Access ME and the file server application.
security_error	The security has been violated. The file server application is not authorized to open a file in the requested access mode, or to write on a read only file.
insufficient_space	There is no room to define or write a new file. The file space allocated to the file server application has been exhausted.
i_o_error	An unrecoverable I/O error has occurred. The file server application is unable to complete an I/O access.
file_does_not_exist	The requested file does not exist.

invalid_file_position	Attempting to position the file beyond the EOI or before the BOI position.
file_service_unavailable	The file type has not been registered or the file type is not available.
protocol_error	The protocol between Dependent File Access ME and Independent File Access ME has been violated.
unexpected_file_close	The file connection between Dependent File Access ME and Independent File Access ME has been broken.
no_seek_on_sequential_file	A seek request cannot be processed on a file that is specified to be of sequential access type.
bad_byte_count	The file length for a read or write request is specified to be zero.
bad_file_name	The file name has illegal characters or has too many characters.
beyond_end_of_file	An attempt to read has gone beyond the end-of-file.
fcb_active	The file control block specified in the service request is already active.
illformatted_request	The request code is invalid.
purge_busy	The file server application is unable to purge the file.
unknown_fcb	The field FCB in the file control block is invalid.
usage_conflict	The file is being used in a different mode by another entity.

```
file_responses = (request_confirmed, request_rejected)

file_requests= (create_file, open_file, delete_file, close_file, write_file,
                read_file, seek_file)

file_size = 0 .. max_byte_file_size

line_number = 0 .. 0ffff(16)

read_file_status = (read_ok, read_eof, line_too_long, access_error)

read_length = 1 .. 0ffff(16)

service_quality = 0 .. 3
```

Command ME 7

Command ME 7

This chapter describes the Command ME. Command ME is the command management software that allows operational control to be distributed throughout the network.

This chapter discusses:

- An overview of the Command ME
- The services required by Command ME
- The services provided by Command ME
- Constants and common types

Overview

This section describes the following:

- Independent and Dependent Command ME.
- The Dependent Command ME services.
- User interfaces to Command ME.
- Command processors which are used by the Command ME.
- The parameter value table which is used by command processors.
- The CLP_SCAN_PARAMETER_LIST procedure which is used by command processors to parse parameter strings.
- The parameter descriptor table which is used by the CLP_SCAN_PARAMETER_LIST procedure.
- Command responses sent by command processors.
- The message templates which are used by command processors to format command responses.
- The type of commands processed by the Command ME.

Components of the Command ME

Command ME consists of the following two components:

- Independent Command ME
- Dependent Command ME

Independent Command ME

The Independent Command ME provides a command interface to network operators. Its main responsibility is to route operator commands to the systems in the catenet where the commands are to be executed. It sends commands to specific systems using Transport layer services or broadcasts commands using the Internet layer services. The Independent Command ME is also responsible for communicating the command responses from the Dependent Command ME back to the operators. (Broadcast commands are supported in a future release).

In the CDCNET NOS environment, the Independent Command ME is implemented as the Operator Support Application (OSA) and also functions as the Independent Alarm ME. Since it depends on a host server application, the Independent Command ME in a NOS environment resides only in a mainframe device interface (MDI) or a mainframe terminal interface (MTI). Figure 7-1 illustrates the Independent Command ME in an NOS environment.

In the NOS/VE environment shown in figure 7-2, the Independent Command ME is implemented solely by the Network Operator Utility which resides on the NOS/VE host.

When an Independent Command ME receives an operator command, it parses the command to determine the requested service. If the operator's command is a request to send a command to a particular DI, the command is further parsed to determine the specified destination. If the specified destination is in the operator's domain of control (domain of control is supported in a future release), the Independent Command ME sends the command to the Dependent Command ME in the specified system. Figure 7-1 illustrates this process.

Since the Independent Command ME has no external program interfaces, it will not be further described in this manual. For more information on network operations and command references, see the CDCNET Network Operations manual.

Dependent Command ME

Every DI in the catenet contains a Dependent Command ME which is responsible for the command execution process within that system. The function of processing a particular command within a DI is divided between the Dependent Command ME and the command processor associated with that command. This chapter describes in detail the features and services of the Dependent Command ME. It also describes the parsing procedures used by the Dependent Command ME and the command processors.

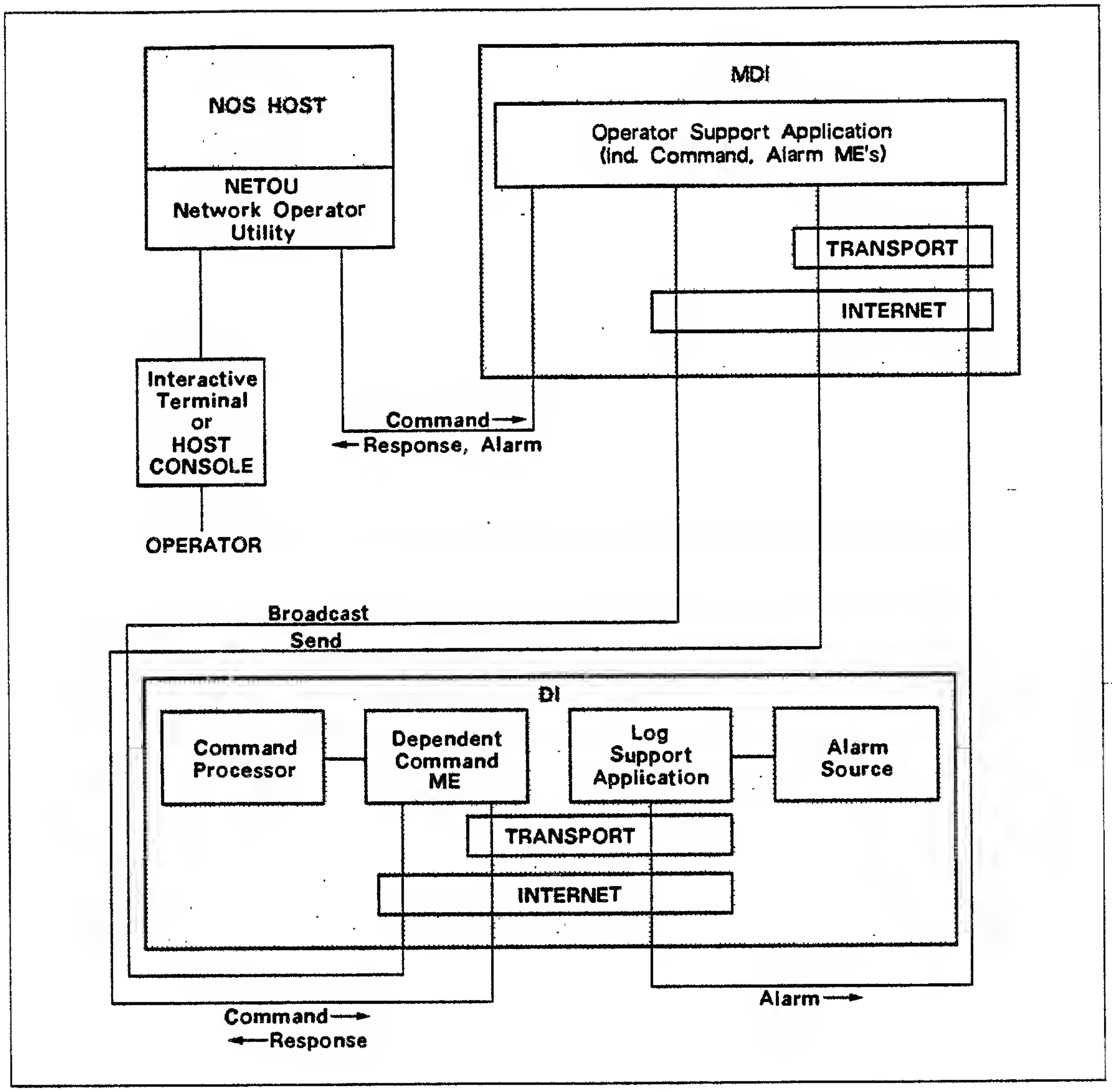


Figure 7-1. Command ME in a NOS Environment

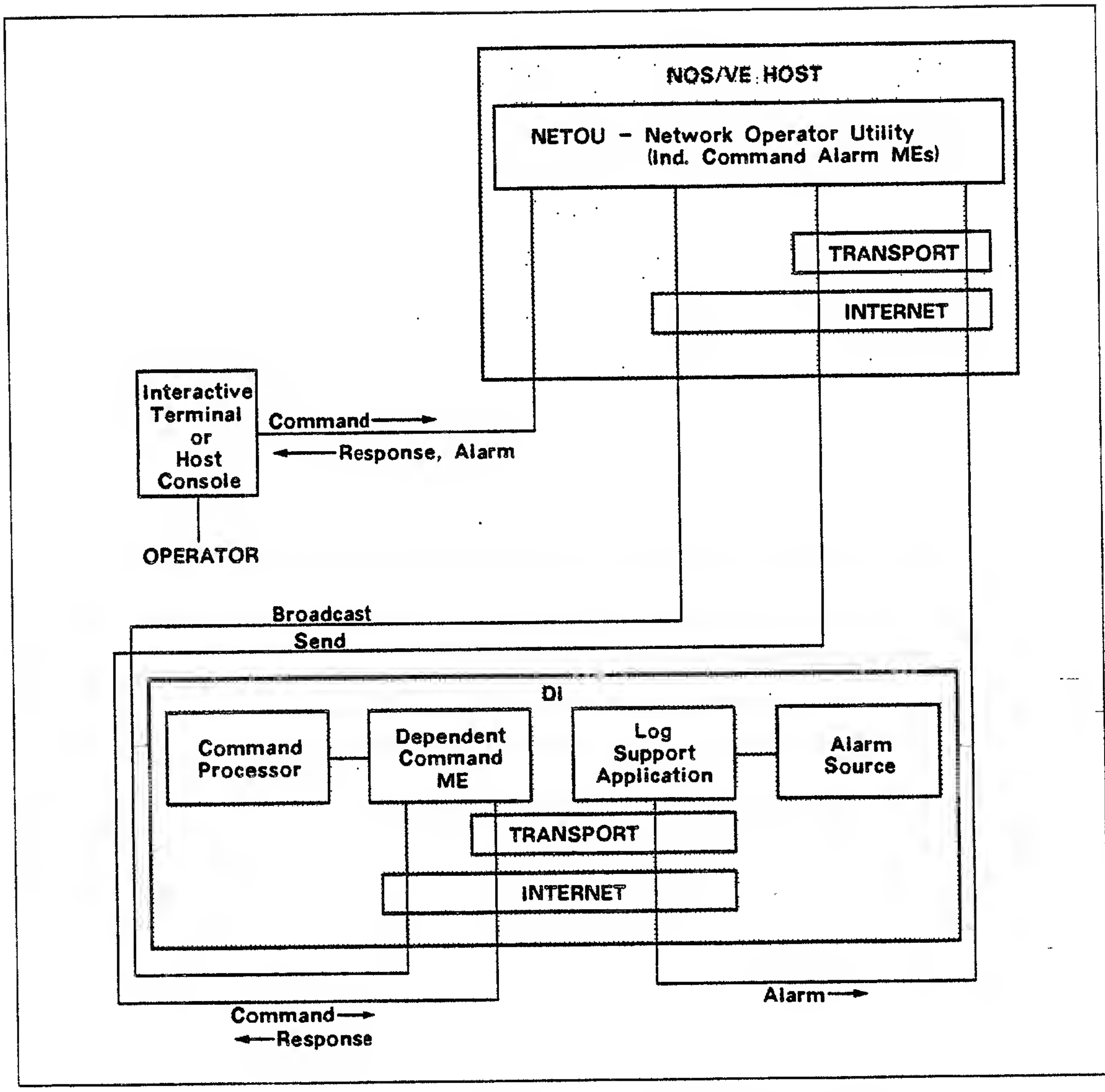


Figure 7-2. Command ME in a NOS/VE Environment

Dependent Command ME Services

The Dependent Command ME provides the following services that are required by all command processors:

- Loading and executing command processors based on requests from the Independent Command ME or another software entity.
- Providing command parsing services to the command processors.
- Providing the means for returning command responses to the Independent Command ME which, in turn, routes the responses back to the originators of the command request.

Dependent Command ME User Interfaces

The Dependent Command ME receives commands through the following interfaces:

- From an Independent Command ME on a transport connection, using Transport layer's services.
- From an Independent Command ME as a datagram or a broadcast datagram, using Internet layer's services.
- From another software component within its own system, using a direct call interface.

As shown in figures 7-1 and 7-2, command data units are always sent by the Independent Command ME using either the Transport or the Internet layer services. Which layer is used depends on how the CDCNET operator sends the command. A SEND_COMMAND, which sends the network command to a specific DI, uses a transport connection. A BROADCAST_COMMAND, which broadcasts the network command to the DIs in the specified community, uses the Internet layer's services. The Dependent Command ME has dedicated Transport layer and Internet layer SAPs through which command data units can be received from the Independent Command ME. For more information on sending commands, see the CDCNET Network Operations Manual.

When a Dependent Command ME receives a command, it parses the command name and requests the Online Loader to load the specified command processor. The Dependent Command ME also verifies that the command is within the operator's command privileges. Command privileges are supported in a future release.

Command Processors

One command processor exists for each command recognized by the system. If an abbreviation for a command is supported, a separate entry point is also defined for it. The abbreviated command entry point, in turn, directly calls the complete command name's entry point. To locate the command processor's entry point address, the characters CMD_ are prefixed by the Dependent Command ME to the command name. For example, the command DISPLAY_NETWORK_STATUS (which displays the status of network solutions connected to the DI), is processed by a command processor with the entry point name of CMD_NETWORK_STATUS.

Every command that is processed by the Dependent Command ME starts as a new task. The task starts once the command processor module is loaded. A command processor task is started with the task attributes defined either by the command processor itself or by a default set of attributes. A command processor defines its task attributes by specifying an externally declared (XDCL) data item of type, TASK_ATTRIBUTE for both the complete and abbreviated command entry point name. The name of the TASK_ATTRIBUTE is the same as the complete or abbreviated name of the entry point except it has a suffix, _TA. Through the task attributes, a command processor can specify the stack size, assign priority, and state if the task is to be preempted.

All network commands are limited to 24 characters, since the prefix CMD_ and the suffix _TA are used to locate the command processors' entry point address and task attributes, respectively. Command ME truncates the name of the command to 24 characters before calling the loader to locate the required names.

The command processor is invoked through a direct call by the Command ME, which also supplies the command processor with a parameter list and a status variable. The command processor uses the parameter list as input information and the status variable as output, to return the command's completion status. The command processor is responsible for parsing the supplied parameters and returning a response to the Dependent Command ME through the status variable. To process the parameter list that accompanies the command, the command processor calls the CLP_SCAN_PARAMETER_LIST procedure which scans the passed parameter list according to the System Command Language (SCL) parameter syntax rules. To scan the parameter list, the CLP_SCAN_PARAMETER_LIST procedure requires a parameter descriptor table that defines the valid parameters for the parameter list. The CLP_SCAN_PARAMETER_LIST procedure and the parameter descriptor table are discussed later in this section.

Parameter Value Table (PVT)

Most command parsing procedures use a parameter value table (PVT) in addition to a command parameter list as input information. The PVT is built by the CLP_SCAN_PARAMETER_LIST procedure, which is described later in this section. The PVT contains an array of parameter name descriptors that are derived from the parameter descriptor table, which is also described later in this section.

CLP_SCAN_PARAMETER_LIST Procedure

Command processors call the CLP_SCAN_PARAMETER_LIST procedure to parse the command parameter string according to SCL parameter syntax rules, and then call the various parameter access procedures to get the actual parameter values. This procedure builds the PVT. The CLP_SCAN_PARAMETER_LIST procedure and the various

parameter access procedures, and the string conversion procedures are described in the Services Provided section of this chapter. The procedures that generate response messages are described in the CDCNET Systems Programmer's Reference Manual, Volume 1.

Parameter Descriptor Table (PDT)

To parse a parameter string, the CLP_SCAN_PARAMETER_LIST procedure requires the parameter list that was passed to the command processor and the PDT. A PDT lists all valid parameter names and a parameter descriptor for each valid parameter. While scanning the parameter list, if the CLP_SCAN_PARAMETER_LIST procedure finds a parameter name which is not defined in the PDT or a parameter specification not allowed by the corresponding parameter descriptor, it returns a standard SCL syntax error in its status variable. The command processor passes the syntax error to the user that entered the command through the status variable passed to the command processor.

Command Responses

Each command processor returns exactly one command response per command request. A command response indicates whether a command was successfully or unsucessfully processed. The command processor passes the response to the Dependent Command ME, and the Dependent Command ME returns the response to the Independent Command ME through the same service (Transport, Internet, or direct call interface) through which the command was received.

The command response consists of a record of type CLT$STATUS which contains a response identifier, a response message data unit, and a status indicator. The status indicator is a boolean value which is set by the command processor to a value of FALSE for an error response and to a value of TRUE for a nonerror response. The Command ME forms a command response protocol data unit by prefixing a header with the response identifier and other information to the command response. All command processors should return response messages to indicate the command processing status. If no response is returned by the command processors, the Dependent Command ME supplies a response indicating that a message was not supplied.

Message Templates

Message templates are used to format command responses and alarm messages that are displayed to the network operator. They are also used to format log messages in reports generated by the Network Performance Analyzer (NPA). Message templates contain constant or fixed-type information that does not change from one instance of a message to the another. Variable information is combined with an appropriate message template to form a display for an end user. For more information on alarms, see chapter 7 in this manual. The CDCNET Network Performance Analyzer Manual has detailed information on network performance reports.

Types of Commands

The following commands are processed by the Command ME:

Configuration commands

Configuration commands contain information to configure or reconfigure a DI. Configuration commands include the following:

- Commands to reconfigure lines, boards, devices, network solutions, and DIs
- Commands to reset systems and effect their reload
- Commands to add or delete community titles

Status commands

Status commands check the current state of element(s) of a system. The status of the physical elements like the communication links, modems, cluster controllers, terminals, channels, and the status of logical elements such as programs, entities, end-users, connections, and SAPs are accessible through the status commands.

Statistics commands

Statistics commands access statistical information maintained for the software components in the DI. The following are the different types of statistical information maintained for physical and logical elements:

- Start and termination time
- Execution time
- CPU overhead information
- Data rates
- Error rates

Log commands

Log commands are used to activate or deactivate the logging of statistical and/or status information. Log commands also identify which log data units are sent as alarms.

For more information on these commands, see the CDCNET Network Operations Manual.

Services Required

The Dependent Command ME relies on the services of other entities to provide services to its own users. This section briefly describes the services of each of these entities.

Internet Layer (3B)

The services of the Internet layer are used to receive and send datagrams carrying commands and command responses. Command ME uses the OPEN_INTERNET_SAP to register its well known SAP with Internet. See chapter 16 in this manual for more information.

Generic Transport Layer (4)

The services of Generic Transport are used to send and receive data carrying command responses and commands repectively. See chapter 14 in this manual for more information.

Online Loader

The services of the Online Loader are used to obtain the entry point for a command processor and, if neccessary, to load the command processor module. See Software Load Process in the CDCNET Systems Programmer's Reference Manual, Volume 1, for more information.

System Ancestor

The services of the System Ancestor are used to initiate the execution of command processor tasks. Command ME is responsible for registering a recovery procedure for each command processor task. The recovery procedure is executed by the System Ancestor when the command processor task aborts. The recovery procedure forces the Command ME to send a response back to the command originator indicating that the command aborted. See Software Load Process in the CDCNET Systems Programmer's Reference Manual, Volume 1, for more information.

Dependent Log ME (LSA)

The services of the Log ME, also known as the Log Support Appplication, are used to log command responses for transport connections that are no longer connected. These services are also used to log other unusual events. See the CDCNET Diagnostics Messages manual for log messages issued by the Command ME.

Statistics Manager

The services of the Statistics Manager are used to open a Command ME statistics SAP and to report Command ME statistics. See the CDCNET Systems Programmer's Reference Manual, Volume 1 for more information.

Status Manager

The services of the Status Manager are used to open a Command ME software status SAP. See the CDCNET Systems Programmer's Reference Manual, Volume 1 for more information.

Services Provided

This section describes the services provided by the Dependent Command ME to its users. Figure 7-3 illustrates all the procedures used by Command ME and the command processors, and groups them according to the various functions they perform. The following pages describe all the procedures illustrated in this figure, except the response message procedures, which are described in the CDCNET Systems Programmer's Reference Manual, Volume 1.

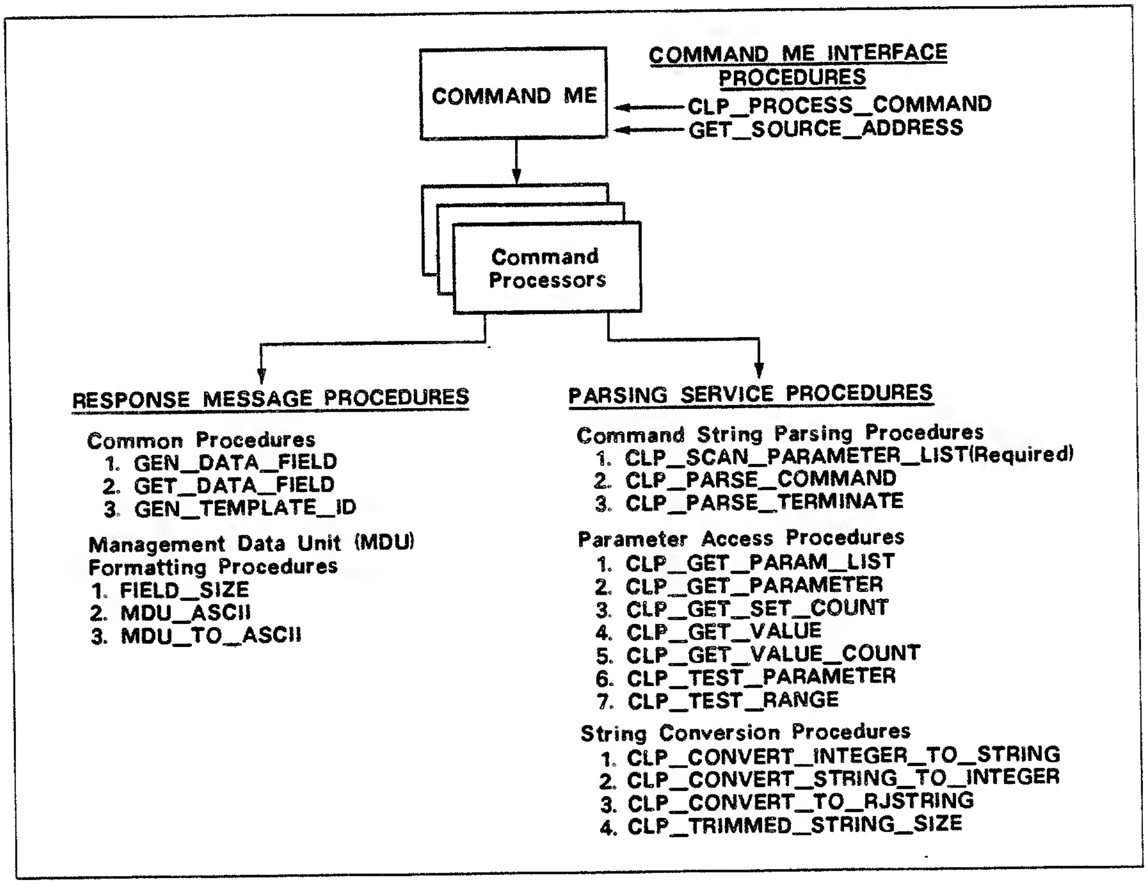


Figure 7-3. Command ME Procedures

The following two parameters are described in detail here because they are common to many procedures described later in this section:

CLT$STATUS	Returns both success and error responses to commands. See table 7-1.
CLT$VALUE	Gives information on parameter values as entered by the operator. See table 7-2.

Table 7-1. Status Information (CLT$STATUS)

Field	Content
normal	The key field that indicates the completion status of a procedure (type BOOLEAN). This parameter contains the following values. TRUE The procedure completed normally. FALSE The procedure completed abnormally.
response_id	The response identifier. Each command response can be referenced by its response identifier as listed in the CDCNET Diagnostics Messages manual. (type MIN_RESPONSE_MESSAGE_ID .. MAX_RESPONSE_MESSAGE_ID)
condition	A pointer to the message buffer which contains the command response text (type BUF_PTR).

Table 7-2. Evaluated Expression Value (CLT$VALUE)

Field	Content
descriptor	The name of the value kind returned (type STRING).
kind	The kind of value returned (type CLC$UNKNOWN_VALUE .. CLC$CCODE_VALUE of). CLC$UNKNOWN_VALUE An unknown value kind. CLC$STRING_VALUE The string value. The STR field contains a pointer to the string record. CLC$NAME_VALUE The name. The NAME field contains the name record. CLC$INTEGER_VALUE The integer value. The INT field contains the integer record. CLC$BOOLEAN_VALUE The boolean value. The BOOL field contains the boolean record. CLC$CCODE_VALUE The CCODE value. The CCCODE field contains the character code value.
str	Pointer to a string record (OST$STRING). This field is generated only if the kind field is set to CLC$STRING_VALUE. **Field** / **Content** size — Actual string length (OST$STRING_SIZE, 0 through OSC$MAX_STRING_SIZE). value — String (256 characters).
name	Name record (CLT$NAME). This field is generated only if the KIND field is set to CLC$NAME_VALUE. **Field** / **Content** size — The actual name length (1 through OST$MAX_NAME_SIZE). value — The name (31 characters).

(Continued)

Table 7-2. Evaluated Expression Value (CLT$VALUE) *(Continued)*

Field	Content
int	Integer record (CLT$INTEGER). This field is generated only if the KIND field is set to CLC$INTEGER_VALUE. **Field** / **Content** value — Integer value (integer). radix — Radix used (2 through 16). radix_specified — Indicates whether a radix was specified. TRUE Radix specified. FALSE Radix omitted.
bool	The boolean record (CLT$BOOLEAN). This field is generated only if the KIND field is set to CLC$BOOLEAN_VALUE. **Field** / **Content** value — A boolean value (boolean). kind — This indicates the keyword used to specify value (CLT$BOOLEAN_KINDS). CLC$TRUE_FALSE_BOOLEAN Value TRUE or FALSE. CLC$YES_NO_BOOLEAN Value YES or NO. CLC$ON_OFF_BOOLEAN Value ON or OFF.
ccode	The character code record (CLT$CCODE). This field is generated only if the kind field is set to CLC$CCODE_VALUE. **Field** / **Content** value — A single octet (0..0FF(16)) value, which represents a keyboard character code. kind — This indicates how the character code parameter was entered. CLC$CCODE_NAME The character code was entered as a name. Example, STX, BEL. CLC$CCODE_STRING The character code was entered as a single character string. Example, 'A', '!'

(Continued)

Table 7-2. Evaluated Expression Value (CLT$VALUE) *(Continued)*

Field	Content
	CLC$CCODE_INTEGER The character code was entered as a number. Example, 01E(16), 30,
	CLC$CCODE_CONTROL The character code was entered using the ^ notation to represent the CTRL key. For example, ^A is the code for the CTRL-A character
str	The actual name or string entered for the parameter. This is present only when the KIND field is CLC$CCODE_NAME or CLC$CCODE_STRING.

The following pages describe the Dependent Command ME procedures and the parsing service procedures.

CLP_PROCESS_COMMAND

This procedure is used for processing commands that are to be executed in the same system as the command originator. As described earlier, the Dependent Command ME can receive commands through direct calls from other software modules within its own system. In other words, this procedure processes commands that are received by the Dependent Command ME through an interface other than an operator interface.

Comdeck CLXPCM

Format CLP_PROCESS_COMMAND (str, status)

Input **str: ost$string**

The command string to be processed.

Output **status: clt$status**

The status record. See table 7-1 for details.

Remarks This procedure accepts a command character string and converts it into the management data unit syntax. The command is then sent to the Dependent Command ME through an intertask message. A command response status variable consisting of a buffer containing the command response data unit, the command response identifier, and the response status flag is returned. In this procedure, control is not returned to the user until a response is returned by the Dependent Command ME. This procedure executes under the user's task and any invalid intertask messages sent to this task while waiting for the command response are lost.

GET_SOURCE_ADDRESS

This procedure is used by a command processor to get the address of the command originator from the Dependent Command ME.

Comdeck MEXGSA

Format GET_SOURCE_ADDRESS (task, source)

Input **task: task_ptr**

The task identifier of a command processor task. If the user supplies a NIL value for this parameter, the task identifier of the task currently executing is used.

Output **source: generic_sap**

The network address of the Independent Command ME which sent the command. See appendix B for a description of the network address.

CLP_SCAN_PARAMETER_LIST

This procedure is used by all command processors for parsing the commands. It scans the parameter list for a command under control of a PDT and parses the parameter list according to the parameter definitions within the specified PDT. It checks to make sure all parameter names within the parameter list are defined in the PDT and that each parameter value is valid for its parameter. If it finds an invalid parameter name or parameter value, it returns an error response in the status variable.

Comdeck **CLXSPL**

Format **CLP_SCAN_PARAMETER_LIST (parameter_list, pdt, pvt, status)**

Input **parameter_list: ost$string**

This parameter specifies the parameter list to be scanned.

pdt: clt$parameter_descriptor_table

The parameter descriptor table.

Output **pvt: clt$parameter_value_table**

The parameter value table.

status: clt$status

The status record. See table 7-1 for more information on this record.

CLP_PARSE_COMMAND

This procedure is used by any software which processes SCL formatted commands to identify a specified command.

Comdeck CLXPCOM

Format **CLP_PARSE_COMMAND (command, name_index, name_size, name, separator, parameter_list, empty_command, status)**

Input **command: string (*)**

The command to be parsed.

Output **name_index: ost$string_index**

The position within the command string where the command name begins. This remains undefined if EMPTY_COMMAND parameter (described later in this section) is TRUE.

name_size: ost$string_size

The number of characters in the command or the size of the command name. This remains undefined if EMPTY_COMMAND parameter (described later in this section) is TRUE.

name: clt$name

The name of the command (in uppercase letters) that is returned. This remains undefined if EMPTY_COMMAND parameter (described later in this section) is TRUE.

separator: clt$lexical_kinds

The separator between the command name and the parameters for the command. This remains undefined if EMPTY_COMMAND parameter (described later in this section) is TRUE. Possible values are:

CLC$SPACE_TOKEN

CLC$COMMA_TOKEN

CLC$EOL_TOKEN

parameter_list: ost$string

A list of the command's parameters. This remains undefined if EMPTY_COMMAND parameter (described later in this section) is TRUE.

empty_command: boolean

This specifies whether the command is empty; that is, it consists of only spaces and/or comments.

TRUE	Command is empty.
FALSE	Command is not empty.

status: clt$status

The status record. See table 7-1 for more information on this record.

CLP_PARSE_TERMINATE

This procedure is used by any software component that directly processes SCL formatted commands. After processing the command, this procedure can be called to free the memory that was allocated to the parameter value table during the parsing of the command. A module which directly processes a command must provide a parameter value table to be used by command string parsing procedures such as the CLP_SCAN_PARAMETER_LIST.

Comdeck **CLXPTRM**

Format **CLP_PARSE_TERMINATE (pvt, status)**

Input **pvt: clt$parameter_value_table**

The parameter value table which contains pointers to all the memory areas that were allocated during the parsing of the command.

Output **status: clt$status**

The status record. See table 7-1 for more information on this record.

CLP_GET_PARAM_LIST

This procedure returns the entire command parameter list, in its uninterpreted form, as a string. If no parameters are given, a null string is returned. The parameter list used is the list scanned by a prior CLP_SCAN_PARAMETER_LIST call.

Comdeck CLXGPL

Format CLP_GET_PARAM_LIST (parameter_list, pvt, status)

Input **pvt: clt$parameter_value_table**

The parameter value table which was returned by the CLP_SCAN_PARAMETER_LIST procedure.

Output **parameter_list: ost$string**

The parameter list record.

status: clt$status

The status record. See table 7-1 for more information on this record.

CLP_GET_PARAMETER

This procedure returns the entire value list for a specified parameter. The parameter list used is the parameter list scanned by a prior CLP_SCAN_PARAMETER_LIST call.

Comdeck CLXGPAR

Format **CLP_GET_PARAMETER (parameter_name, pvt, value_list, status)**

Input

parameter_name: string (*)

One of the parameter names for the specified parameter.

pvt: clt$parameter_value_table

The parameter value table which was returned by the CLP_SCAN_PARAMETER_LIST procedure.

value_list: ost$string

The parameter's value list record.

status: clt$status

The status record. See table 7-1 for more information on this record.

CLP_GET_SET_COUNT

This procedure determines the number of value sets supplied for a particular parameter. If the parameter was not supplied, a value set count of zero is returned. A value set is a set of values enclosed in parentheses specified for a parameter. The parameter list used is the parameter list scanned by a prior CLP_SCAN_PARAMETER_LIST call.

Comdeck CLXGSC

Format **CLP_GET_SET_COUNT (parameter_name, pvt, value_set_count, status)**

Input

parameter_name: string (*)

One of the parameter names for the specified parameter.

pvt: clt$parameter_value_table

The parameter value table which was returned by the CLP_SCAN_PARAMETER_LIST procedure.

Output

value_set_count: 0 .. clc$max_value_sets

The number of value sets given for the parameter.

status: clt$status

The status record. See table 7-1 for more information on this record.

CLP_GET_VALUE

This procedure returns a parameter value that was given, in the parameter list. The parameter list used is the list scanned by a prior CLP_SCAN_PARAMETER_LIST call.

Comdeck CLXGVAL

Format **CLP_GET_VALUE (parameter_name, pvt, value_set_number, value_number, low_or_high, value, status)**

Input

parameter_name: string (*)

One of the parameter names for the specified parameter.

pvt: clt$parameter_value_table

The parameter value table which was returned by the CLP_SCAN_PARAMETER_LIST procedure.

value_set_number: 1 .. clc$max_value_sets

The value set number indicating which value set of the PARAMETER_NAME is being referenced. (The number of value sets for the PARAMETER_NAME is returned using the CLP_GET_SET_COUNT procedure.)

value_number: 1 .. clc$max_values_per_set

The value number indicating which value of the VALUE_SET_NUMBER is being referenced. (The number of values in the VALUE_SET parameter is returned using the CLP_GET_VALUE_COUNT procedure.)

low_or_high: clt$low_or_high

Indicates whether the upper or lower bound of the range is returned.

CLC$LOW	Return the lower bound.
CLC$HIGH	Return the upper bound.

Output

value:clt$value

The parameter value. See table 7-2 for more information on this record.

status: clt$status

The status record. See table 7-1 for more information on this record.

Remarks If the parameter list of the command did not specify a value for the parameter specified on the CLP_GET_VALUE call, CLP_GET_VALUE returns value kind CLC$UNKNOWN_VALUE in the value record returned.

CLP_GET_VALUE_COUNT

This procedure returns the number of values specified in a value set.

Comdeck CLXGVC

Format CLP_GET_VALUE_COUNT (parameter_name, pvt, value_set_number, value_count, status)

Input

parameter_name: string (*)

One of the parameter names for the specified parameter.

pvt: clt$parameter_value_table

The parameter value table which was returned by the CLP_SCAN_PARAMETER_LIST procedure.

value_set_number: 1 .. clc$max_value_sets

The value set number.

Output

value_count: 0 .. clc$max_values_per_set

This specifies the number of values given in the specified value set for the specified parameter.

status: clt$status

The status record. See table 7-1 for more information on this record.

CLP_TEST_PARAMETER

This procedure tests whether a parameter list contains a value for a specified parameter. The parameter list used is the parameter list scanned by a prior CLP_SCAN_PARAMETER_LIST call.

Comdeck CLXTPAR

Format **CLP_TEST_PARAMETER (parameter_name, pvt, parameter_specified, status)**

Input

parameter_name: string (*)

One of the parameter names for the specified parameter.

pvt: clt$parameter_value_table

The parameter value table which was returned by the CLP_SCAN_PARAMETER_LIST procedure.

Output

parameter_specified: boolean

Indicates whether the parameter is specified.

TRUE	Parameter is specified.
FALSE	Parameter is omitted.

status: clt$status

The status record. See table 7-1 for more information on this record.

CLP_TEST_RANGE

This procedure determines whether a particular value for a particular parameter was specified as a range. The parameter list used is the parameter list scanned by a prior CLP_SCAN_PARAMETER_LIST call.

Comdeck CLXTRNG

Format **CLP_TEST_RANGE (parameter_name, pvt, value_set_number, value_number, range_specified, status)**

Input

parameter_name: string (*)

One of the parameter names for the specified parameter.

pvt: clt$parameter_value_table

The parameter value table which was returned by the CLP_SCAN_PARAMETER_LIST procedure.

value_set_number: 1 .. clc$max_value_sets

The value set number.

value_number: 1 .. clc$max_values_per_set

The value number.

Output

range_specified: boolean

This parameter indicates whether the value is a range.

TRUE	The value is specified as a range.
FALSE	The value is not specified as a range.

status: clt$status

The status record. See table 7-1 for more information on this record.

CLP_CONVERT_INTEGER_TO_STRING

This procedure converts an integer to its string representation in the specified radix.

Comdeck CLXCI2S

Format **CLP_CONVERT_INTEGER_TO_STRING (int, radix, include_radix_specifier, str, status)**

Input

int: integer

The integer value that is to be converted.

radix: 2 .. 16

The radix in which the integer's value is to be represented (2 through 16).

include_radix_specifier: boolean

This parameter indicates whether a trailing radix enclosed in parentheses is included.

TRUE	The radix is included.
FALSE	The radix is omitted.

Output

str: ost$string

The string record.

status: clt$status

The status record. See table 7-1 for more information on this record.

Remarks

- If requested, a trailing radix enclosed in parentheses is included in the string.
- If the integer is negative, a minus sign is included as the leftmost character in the string.
- If the specified radix is greater than 10 and the leftmost digit of the result is greater than 9, a leading 0 digit is added. For example, if the integer value is 240 (decimal), the radix is 16, and the allocated string is three characters, the string representation is 0F0.

CLP_CONVERT_STRING_TO_INTEGER

This procedure converts the string representation of an integer to the integer value. The string representation can include a leading sign and a trailing radix enclosed in parentheses.

Comdeck CLXCS2I

Format **CLP_CONVERT_STRING_TO_INTEGER (str, int, status)**

Input **str: string (*)**

The string to be converted.

Output **int: clt$integer**

This specifies the converted integer value along with the radix in which the integer was represented.

Field	Content
value	The integer value (type integer).
radix	The representation radix (2 through 16).
radix_ specified	This indicates whether a radix was specified in the string (type boolean). TRUE The radix was specified. FALSE The radix was omitted.

status: clt$status

The status record. See table 7-1 for more information on this record.

CLP_CONVERT_TO_RJSTRING

This procedure converts an integer to its right-justified string representation in the specified radix.

Comdeck CLXCIRS

Format **CLP_CONVERT_TO_RJSTRING (int, radix, include_radix_specifier, fill_character, str, status)**

Input

int: integer

The integer value that is to be converted.

radix: 2 .. 16

The radix in which the integer's value is to be represented (2 through 16).

include_radix_specifier: boolean

This parameter indicates whether a trailing radix enclosed in parentheses is included.

TRUE	The radix is included.
FALSE	The radix is omitted.

fill_character: char

The character used to fill in the unused portions of the returned string.

Output

str: string (*)

The converted string which is right justified.

status: clt$status

The status record. See table 7-1 for more information on this record.

Remarks

- If requested, a trailing radix enclosed in parentheses is included in the string.
- If the integer is negative, a minus sign is included in the string. Its position within the string depends on the fill character used. If the fill character is a space, the minus sign is positioned immediately before the first digit. If the fill character is not a space, the minus sign is the leftmost character in the string.
- If the specified radix is greater than 10 and the leftmost digit of the result is greater than 9, a leading 0 digit is added if space for the digit is available in the allocated string. For example, if the integer value is 240 (decimal), the radix specified is 16, and the string allocated is three characters, the string representation is 0F0.

CLP_TRIMMED_STRING_SIZE

This function determines the size of a string after the trailing space characters are removed.

Comdeck CLXTSS

Format **trimmed_string_size:= CLP_TRIMMED_STRING_SIZE (str)**

Input **str: string (*)**

The string for which the trimmed size is returned.

Output **trimmed_string_size: ost$string_size**

The variable to which the size of the string is returned.

Constants and Common Types

This section lists all the constants and type declarations used by the procedures described in this chapter. Each type or constant is preceded by the name of the common deck in which it is defined.

Constants

Common Deck CLDPMAX

```
clc$max_keyword_values = 255
clc$max_parameters = 255
clc$max_parameter_names = 255
clc$max_parameter_values = 255
clc$max_values_per_set = 255
clc$max_value_sets = 255
```

Common Deck OSDNAME

```
osc$max_name_size = 31,
osc$null_name = '                               ';
```

Common Deck CLDSTAT

```
osc$status_parameter_delimiter = CHR (31)
```

Common Types

Common Deck CLDBOOL

```
clt$boolean = record
  value: boolean,
  kind: clt$boolean_kinds,
recend

clt$boolean_kinds = (clc$true_false_boolean, clc$yes_no_boolean,
                     clc$on_off_boolean)
```

Common Deck CLDCCOD

```
clt$ccode = record
  value: 0 .. 0ff(16)
  kind: clt$ccode_kinds
  str: string(3)
recend

clt$ccode_kinds = (clc$ccode_name, clc$ccode_string, clc$ccode_integer,
                   clc$ccode_control)
```

Common Deck CLDPVT

```
clt$how_parameter_given = (clc$omitted_parameter, clc$defaulted_parameter,
                           clc$actual_parameter)
```

Common Deck CLDINT

```
clt$integer = record
  value: integer,
  radix: 2 .. 16,
  radix_specified: boolean,
recend
```

Common deck CLDLEX

```
clt$lexical_kinds = (clc$unknown_token, clc$space_token, clc$eol_token,
  clc$dot_token, clc$semicolon_token, clc$colon_token, clc$lparen_token,
  clc$lbracket_token, clc$lbrace_token, clc$rparen_token,
  clc$rbracket_token, clc$rbrace_token, clc$uparrow_token,
  clc$rslant_token, clc$query_token, clc$comma_token, clc$ellipsis_token,
  clc$exp_token, clc$add_token, clc$sub_token, clc$mult_token,
  clc$div_token, clc$cat_token, clc$gt_token, clc$ge_token, clc$lt_token,
  clc$le_token, clc$eq_token, clc$ne_token, clc$string_token,
  clc$name_token, clc$integer_token, clc$ccode_token)
```

Common Deck CLDPMAX

```
clt$low_or_high = (clc$low, clc$high)
```

Common Deck CLDNAME

```
clt$name = record
  size: ost$name_size,
  value: ost$name,
recend
```

Common Deck CLDPDT

```
clt$parameter_descriptor = record
  required_or_optional: clt$required_or_optional,
  min_value_sets: 1 .. clc$max_value_sets,
  max_value_sets: 1 .. clc$max_value_sets,
  min_values_per_set: 1 .. clc$max_values_per_set,
  max_values_per_set: 1 .. clc$max_values_per_set,
  value_range_allowed: (clc$value_range_not_allowed,
    clc$value_range_allowed),
  value_kind_specifier: clt$value_kind_specifier,
recend

clt$parameter_descriptor_table = record
  names: ^array [1 .. * ] of clt$parameter_name_descriptor,
  parameters: ^array [1 .. * ] of clt$parameter_descriptor,
recend

clt$parameter_name_descriptor = record
  name: ost$name,
  number: 1 .. clc$max_parameters,
recend
```

Common Deck CLDPVT

```
clt$parameter_value_table = record
  case built: boolean of
  = TRUE =
    parameter_list: ^string ( * ),
    names: ^clt$pvt_names,
    parameters: ^clt$pvt_parameters,
    values_area: ^clt$pvt_values_area,
    values: ^clt$pvt_values,
  casend,
recend
```

Common Deck CLDPVT

```
clt$pvt_name = clt$parameter_name_descriptor,
clt$pvt_names = array [1 .. * ] of clt$pvt_name

clt$pvt_parameter = record
  how_given: clt$how_parameter_given,
  case value_set_count: 0 .. clc$max_value_sets of
  = 1 .. clc$max_value_sets =
    first_value_index: 1 .. clc$max_parameter_values,
    last_value_index: 1 .. clc$max_parameter_values,
    value_list_index: ost$string_index,
    value_list_size: ost$string_size,
    name_index: 0 .. clc$max_parameter_names,
  casend,
recend

clt$pvt_parameters = array [1 .. * ] of clt$pvt_parameter

clt$pvt_value = record
  value_set_number: 1 .. clc$max_value_sets,
  value_number: 1 .. clc$max_values_per_set,
  low_or_high: clt$low_or_high,
  value: clt$value,
recend

clt$pvt_values = array [1 .. * ] of clt$pvt_value

clt$pvt_values_area = SEQ ( * )
```

Common Deck CLDSTAT

```
clt$status = record
  normal: boolean,
  response_id: min_response_message_id .. max_response_message_id,
  condition: buf_ptr, { management data unit syntax }
recend
```

Common Deck CLDVAL

```
clt$value = record
  descriptor: string (osc$max_name_size),
  case kind: clc$unknown_value .. clc$ccode_value of
  = clc$unknown_value =
    ,
  = clc$string_value =
    str: ^ost$string,
  = clc$name_value =
    name: clt$name,
  = clc$integer_value =
    int: clt$integer,
  = clc$boolean_value =
    bool: clt$boolean,
  = clc$ccode_value =
    ccode: clt$ccode,
  casend,
recend
```

Common Deck CLDVLK

```
clt$value_kinds = (clc$unknown_value, clc$name_value, clc$integer_value,
                   clc$boolean_value, clc$any_value, clc$ccode_value)
```

Common Deck OSDSTRD

```
osc$max_string_size = 256

ost$string = record
  size: ost$string_size,
  value: string (osc$max_string_size),
recend

ost$string_index = 1 .. osc$max_string_size + 1

ost$string_size = 0 .. osc$max_string_size
osc$max_string_size = 256
```

Common Deck OSDNAME

```
ost$name = string (osc$max_name_size)

ost$name_size = 1 .. osc$max_name_size
```

Log ME 8

Log ME 8

The CDCNET Log Management Entity (ME) is responsible for recording log messages received from all CDCNET software components. Log messages are used to determine the performance of the network. These messages are written on a log file that resides in a host computer system. Any message can be sent as a log message and/or as an alarm. See chapter 9 for information on Alarm ME.

The Log ME consists of two components:

- Independent Log ME
- Dependent Log ME

Independent Log ME

The Independent Log ME receives log data units from a Dependent Log ME and writes them on a log file that is stored on the host computer system. At least one DI or a NOS/VE host in a catenet is required to have an Independent Log ME. The Independent Log ME resides only in systems with access to permanent storage. Only Mainframe Device Interfaces (MDI) and Mainframe Terminal Interfaces (MTI) that are connected to hosts running NOS (Network Operating System) are capable of being configured with an Independent Log ME. In a NOS environment (see figure 8-1), the Independent Log ME is implemented by a software component that resides in an MDI/MTI connected to a mainframe running NOS. In a NOS/VE environment, the Independent Log ME resides on the host itself (see figure 8-2). The Independent Log ME forwards the log data units to a host program called the Network Log Server (NETLS), which is responsible for physically writing the log messages (log data units) to mass storage.

This chapter does not further describe the Independent Log ME, because there are no interfaces to the Independent Log ME that are accessible to an external user, such as a gateway program.

Dependent Log ME

The Dependent Log ME generates log data units for its local users. Log data units are transmitted to one or more Independent Log MEs, to be written to a log file. The Dependent Log ME and the Dependent Alarm ME are implemented together in a software component called the Log Support Application.

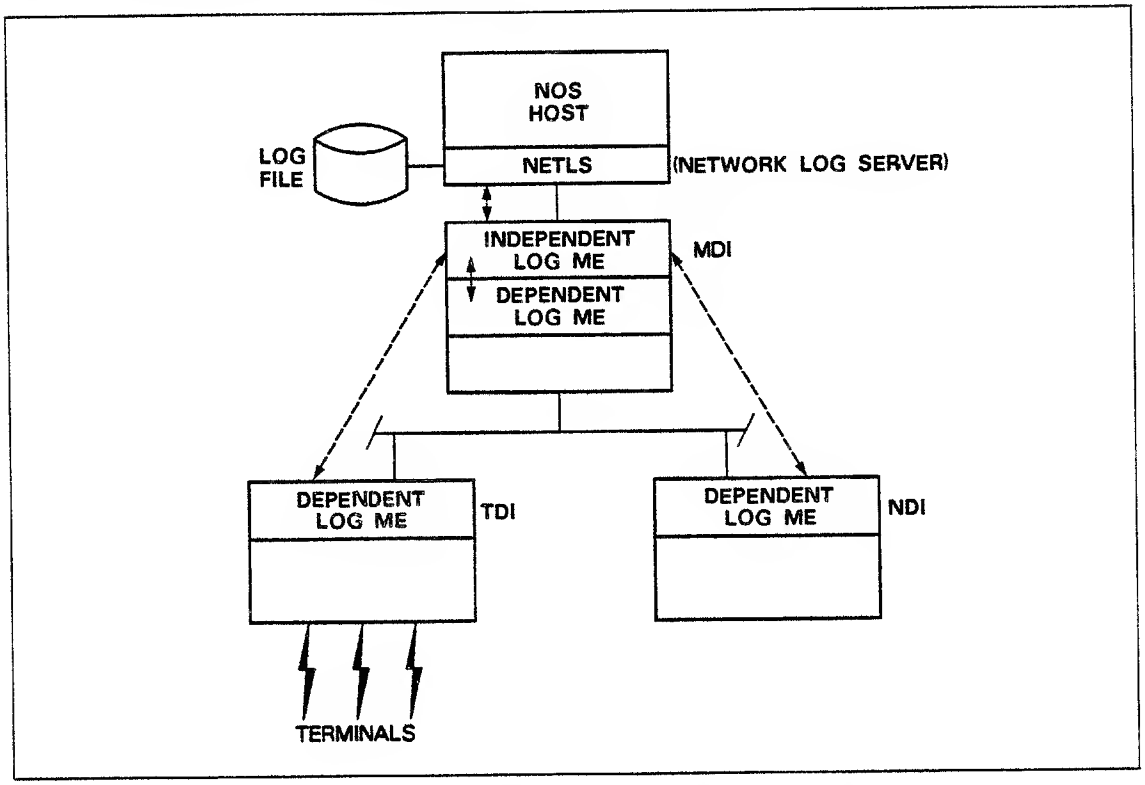


Figure 8-1. Log ME in a NOS Environment

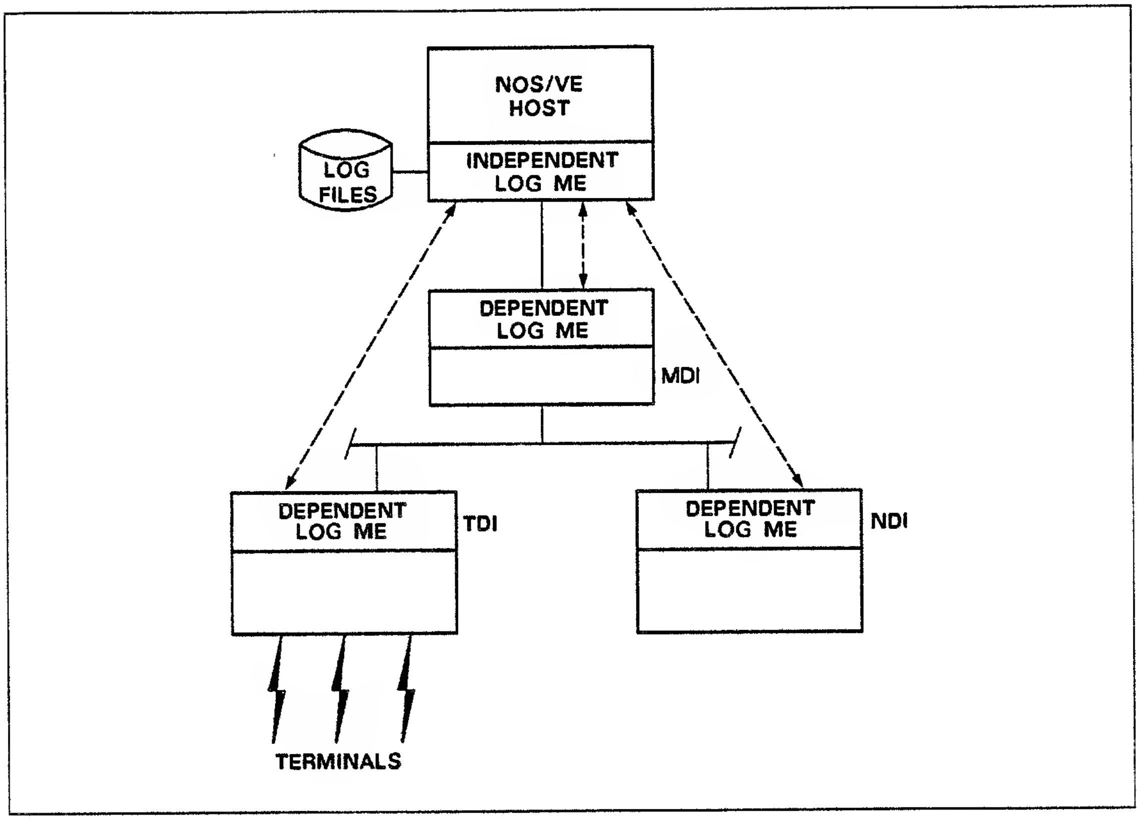


Figure 8-2. Log ME in a NOS/VE Environment

Log Messages

Log messages provide valuable network performance information. Software components in a DI collect information about themselves and this information is used for the following tasks:

- Account for resource usage
- Determine how system resources are being used
- Provide performance and throughput information for site management
- Identify actual and potential software and hardware failures
- Determine if the network is running properly
- Identify potential bottlenecks
- Report the current hardware/software configuration
- Provide input for designing future enhancements to CDCNET

If a site needs information to perform the above mentioned tasks, it can, through configuration commands, have that information generated and reported in the form of log messages. These log messages are then stored on a log file in the host computer system for later analysis with a tool called the Network Performance Analyzer (NPA). For more information on the Network Performance Analyzer, see the CDCNET Network Performance Analyzer Manual. Logging configuration commands are given in the CDCNET Network Configuration and Site Administration Guide.

A log message is identified by a log message identifier and is made up of the following two parts:

- A fixed part
- A variable part

The fixed part of the message is constant and does not change from one instance of the log message to another and is defined by one or more message templates which reside in the host computer system. A message template is a string of text characters and control sequences. The control sequences specify the actions required to combine the text with the variable part of the message to produce a displayable message.

The variable part of the message is provided by the user who is generating the log message. The user provides the variable part of the log message along with the template identifiers that identify the message templates to be used for each message. The template identifiers and variable part of the message are provided in the management data unit (MDU) format. The procedure GEN_TEMPLATE_ID is used to generate MDUs containing the template identifiers and the GEN_DATA_FIELD procedure is used to generate MDUs containing the variable part. Both these procedures are described in the CDCNET Systems Programmer's Reference Manual, Volume 1.

The Dependent Log ME takes the information provided by the user and adds headers with the appropriate system information (date, time, system address, and system name). If the message is selected to be logged, the Dependent Log ME sends the message to

the Independent Log ME, which writes the message onto a log file. In a NOS environment, the NETLS helps the Independent Log ME to write the message to a log file.

The NPA uses the template identifier provided by the user to get the appropriate message template, and combines it with the variable part provided by the user to make a complete log message.

Network Operator Interface

The network operator interface, shown in figure 8-3 as the command processor interface, provides the network operator with the capability to control the logging environment. Commands to define the logging environment can be placed either in the configuration file or entered through an operator interface while the network is running. These commands allow the site to specify which of the CDCNET-defined messages are to be written to a log file and which messages are to be sent as alarms.

Each site has the capability to define or group together subsets of DIs within the catenet and have all systems within this subset send their log messages to a common log file. This subset is referrred to as a log group. Each DI is configured with the name(s) of the log groups it belongs to, along with the message numbers of the log messages it is to log. For the initial releases, only one log group called CATENET is supported. CATENET includes all the DIs in the catenet.

For more information on configuration commands, see the CDCNET Configuration and Site Administration Guide. Information on the network operator interface is provided in the CDCNET Network Operations Manual.

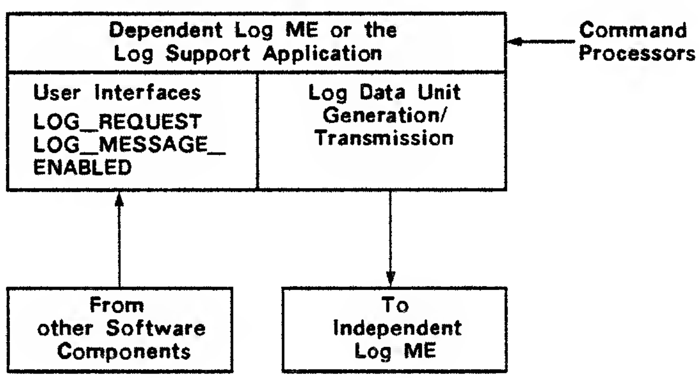


Figure 8-3. Log ME Interfaces

User Interfaces to Dependent Log ME

As shown in figure 8-3, there are two common interfaces to the Dependent Log ME which are used by all CDCNET software components:

LOG_MESSAGE_ENABLED
: This interface allows a user to determine if a specific log message is selected to be logged and/or sent as an alarm. The user provides the log identifier and a boolean flag is returned indicating whether or not the message is enabled. The value is normally set to TRUE if the message is enabled or to FALSE if the message is disabled. However, if memory or buffer space in the system is low, the flag may be set to FALSE even if the message is enabled. The decision is based on the state of the memory and buffer space, and the log message priority. With a FALSE value, the user should generate a message, thus conserving system resources.

LOG_REQUEST
: This interface allows the user to send a message that has been selected to be logged to an Independent Log ME. It is also used to send messages selected as alarms to an Independent Alarm ME. The user provides a buffer containing the template identifiers, the variable parts of the message, and the log identifier. The user does not know if the message is to be logged and/or sent as an alarm. The Log Support Application determines where the message is to be sent, based on operator commands, and forwards it to the appropriate destination(s).

The CYBIL definitions of these interfaces are given in the Services Provided section, later in this chapter.

Services Required

For the Dependent Log ME to provide services to its own users it, in turn, depends on the services of other software components. Brief descriptions of the services of each of these software components follow.

Generic Transport

Generic Transport services provide the transport connections needed to transmit the log data units to the appropriate Independent Log ME(s). See chapter 14 in this manual for details.

Directory ME

Directory ME is used by both the Independent and Dependent Log MEs. Both Independent and Dependent Log MEs use the Directory ME's translation services to locate the address of the local Generic Transport so that they can open a Generic Transport SAP and communicate with each other. The Independent Log ME also registers its title with the Directory ME so that the Dependent Log ME can locate its address. The Dependent Log ME uses the Directory ME's title translation services to obtain the Independent Log ME's address. See chapter 5 in this manual for more information.

Statistics Manager

The Dependent Log ME uses the services of the Statistics Manager to generate statistics such as the number of log messages generated from a specific system, the number of discarded log messages, and so on.

The Dependent Log ME opens a statistics SAP, collects the statistics, and provides a procedure which is used by the Statistics Manager to report statistics. See the CDCNET Systems Programmer's Reference Manual, Volume 1, for more information on the Statistics Manager.

Services Provided

This section defines the external services provided by the Dependent Log ME to its users. It describes the service requests exchanged between the user and the Log ME. Constants and common types are given at the end of this chapter.

Before generating a log message, the LOG_MESSAGE_ENABLED function is used to determine if the message has been selected for logging and/or as an alarm for the local system. If it is selected, the user generates the log message, and the LOG_REQUEST procedure is used to send the message to be written on a log file and/or delivered to the network operator as an alarm.

Users include the common deck LSXLOGR in their calling modules. This deck contains all the externally referenced (XREF) declarations for the Dependent Log/Alarm ME interfaces. The following pages describe these two interfaces.

LOG_MESSAGE_ENABLED Function

The LOG_MESSAGE_ENABLED function is used by all the software components in a local CDCNET system to determine if a log message is enabled (defined as a log message and/or an alarm) and is to be generated.

CDCNET configuration and network operation commands define which messages are to be logged and which messages are to be sent to the network operator as an alarm.

Comdeck **LSXLOGR**

Format **message_enabled: = LOG_MESSAGE_ENABLED (log_message_id, priority)**

Input **log_message_id: log_msg_id_type**

A unique 16-bit integer which identifies each log message in a CDCNET system.

priority: log_priority

This parameter indicates the priority of the log message and will be supported in a future release.

Output **log_message_enabled: boolean**

This parameter indicates whether or not the message is to be generated. If the log message is enabled (defined as a log message and/or as an alarm), this function returns a TRUE value and the user generates the associated log message and calls the LOG_REQUEST procedure to send the message.

If this function returns a FALSE value, the log message is not enabled and the user does not generate the log message. However, if memory or buffer space in the system is low, a FALSE value will be returned even if the message is enabled. With a FALSE value, the log message is not to be generated and system resources are conserved. This parameter contains one of the following values:

TRUE	Message is to be generated.
FALSE	Message is not to be generated.

LOG_REQUEST Procedure

The LOG_REQUEST procedure is used by all software components in the local CDCNET system to send a log message to the Log Support Application. CDCNET network operator commands define which messages are to be logged and which messages are to be sent to the network operator as alarms. The Log Support Application (Dependent Log/Alarm ME) determines if the log message is currently defined as a log message and/or as an alarm, and forwards the message to the appropriate destination.

Comdeck **LSXLOGR**

Format **LOG_REQUEST (log_message_id, log_message)**

Input **log_message_id: log_msg_id_type**

A unique 16-bit integer which identifies each log message in a CDCNET system.

log_message: buf_ptr

A pointer to the buffer containing the template identifiers and the variable parts of the log message in MDU format. Each message is made up of a fixed text and a variable text. Message templates define the fixed text and the actions required to combine the text with the variable parts of the message, thereby producing a displayable message. (The procedure GEN_TEMPLATE_ID is used to generate MDUs containing the template identifiers, and the GEN_DATA_FIELD procedure is used to generate MDUs containing the variable parts. Both these procedures are described in the CDCNET Systems Programmer's Reference Manual, Volume 1.)

If the message contains no variable parts, the buffer must not be left empty; it must contain one or more template identifiers defining the fixed part of the message.

Constants and Common Types

This section lists the constants and common types used in the Log ME's service requests. These data types are defined in the common decks CMEECCR and LSDALDS.

Constants

Common deck CMEECCR

```
min_log_message_id = 0
max_log_message_id = 32999
```

Common Types

Common deck CMEECCR

```
log_msg_id_type = min_log_message_id ..
max_log_message_id
```

Common deck LSDALDS

```
log_priority = (log_critical, log_high, log_medium, log_low)
```

Alarm ME 9

Alarm ME 9

The CDCNET Alarm Management Entity (ME) is responsible for displaying alarms received from all CDCNET software components. Alarms are basically log messages that are displayed at the NOS host console (NAM K-Display) or at the operator's remote terminal, and/or written to a mass storage file. Any message can be sent as either a log message and/or as an alarm. Alarms are used to alert the operator to any problems in the network. This chapter gives a brief overview of the Alarm ME. For complete information on Log and Alarm MEs, see chapter 8 in this manual.

The Alarm ME consists of two components:

- Independent Alarm ME
- Dependent Alarm ME

Independent Alarm ME

The Independent Alarm ME resides only in systems with access to mass storage, because the message templates needed to produce a displayable message reside on a mass storage device. Only Mainframe Device Interfaces (MDI) and Mainframe Terminal Interfaces (MTI) that are connected to hosts running NOS (Network Operating System) are capable of being configured with an Independent Alarm ME. In the NOS/VE environment (Network Operating System/ Virtual Environment), Independent Alarm ME resides on the host itself as part of a host program called the Network Operator Utility (NETOU) (see figure 9-2). At least one CDCNET system in a catenet is required to have an Independent Alarm ME.

In a NOS environment (see figure 9-1), the Independent Alarm and Independent Command MEs are implemented together in a software component known as the Operator Support Application (OSA). OSA resides in an MDI/MTI connected to a mainframe running NOS. The Independent Alarm ME forwards the alarm data units the NETOU. NETOU is responsible for combining message templates (which reside on the host) with the variable information (provided by the software component generating the message) to form an alarm. The alarm is forwarded to each operator whose domain includes the system from which the alarm originated.

This chapter does not further describe the Independent Alarm ME, because there are no program interfaces to the Independent Alarm ME that are accessible to a gateway program.

Dependent Alarm ME

The Dependent Alarm ME generates alarm data units for its local users. Alarm data units are transmitted to one or more Independent Alarm MEs to be displayed at an operator's console. The Dependent Log ME and the Dependent Alarm ME are implemented together in a software component called the Log Support Application.

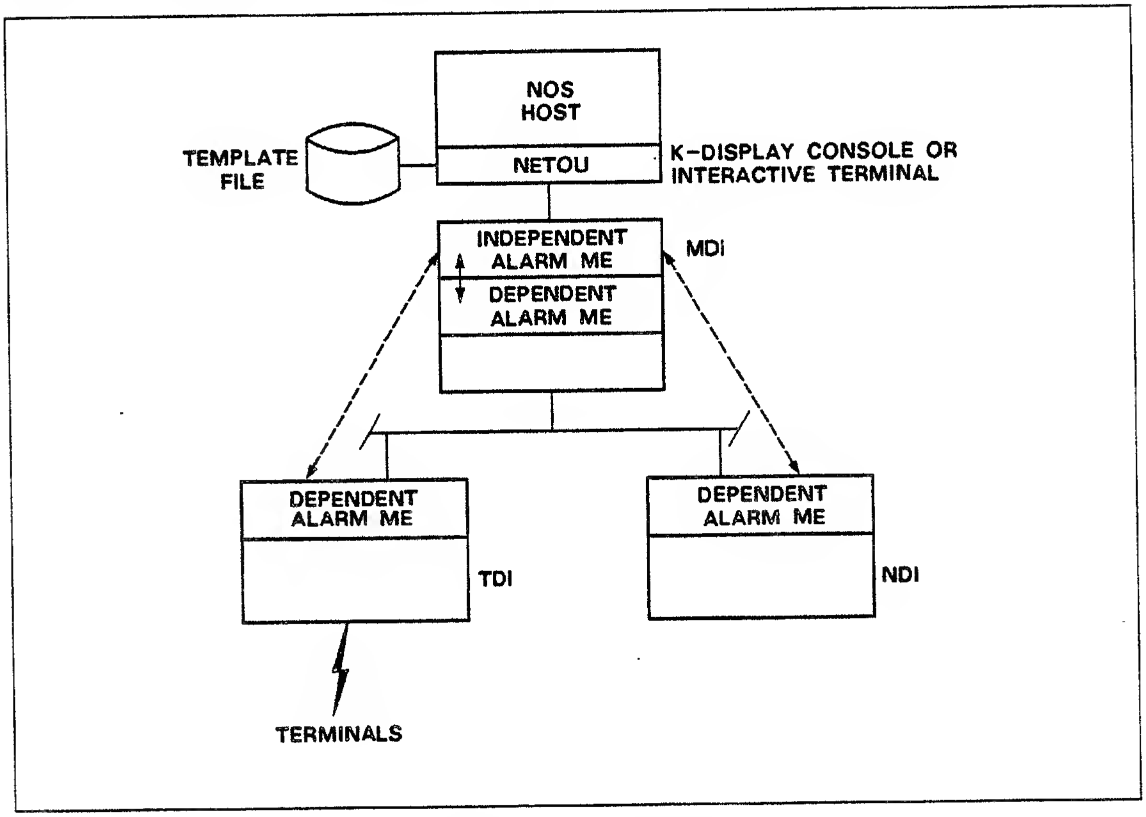


Figure 9-1. Alarm ME in a NOS Environment

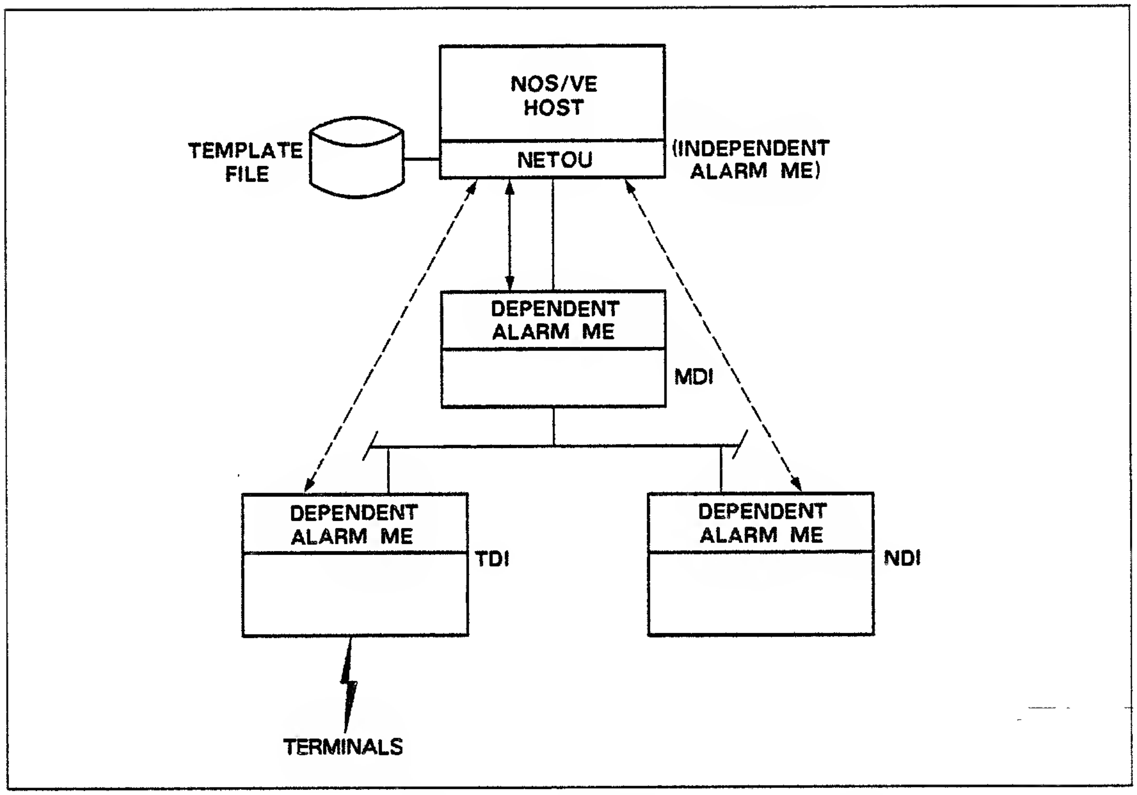


Figure 9-2. Alarm ME in a NOS/VE Environment

Network Operator Interface

The network operator interface, shown in figure 9-3 as the command processor interface, provides the network operator with the capability to control the alarm environment. Commands to define the alarm environment can be placed either in the configuration file or entered through an operator interface while the network is running. These commands allow each site to specify which of the CDCNET-defined messages are to be written to a log file and which messages are to be sent as alarms.

For more information on configuration commands, see the CDCNET Configuration and Site Administration Guide. Information on network operator interface is provided in the CDCNET Network Operations Manual.

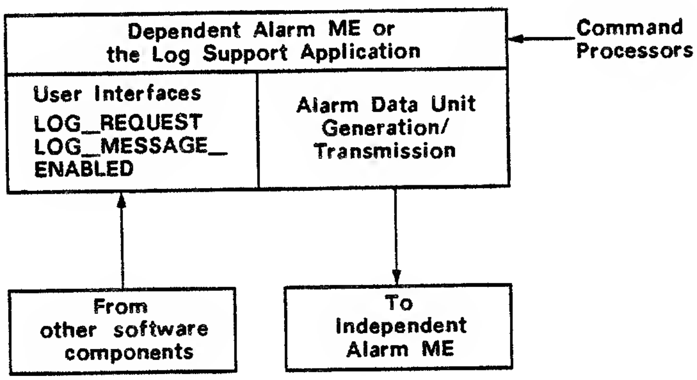


Figure 9-3. Alarm ME Interfaces

User Interfaces to Dependent Alarm ME

As shown in figure 9-3, there are two common interfaces to the Dependent Alarm and Log ME which are used by all CDCNET software components. Configuration commands determine if a message is to be enabled as an alarm and/or log message. Whether it is an alarm or log message is transparent to the Log and Alarm ME users. See chapter 8 in this manual for details on these interfaces and their CYBIL definitions.

Services Required

For the Dependent Alarm ME to provide services to its own users it, in turn, depends on the services of other software components. Brief descriptions of the services of each of these software components follow.

Generic Transport

Generic Transport services provide the transport connections needed to transmit the alarm data units to the appropriate Independent Alarm ME(s). See chapter 14 in this manual for details.

Directory ME

Directory ME is used by both the Independent and Dependent Alarm MEs. Both Independent and Dependent Alarm MEs use the Directory ME's translation services to locate the address of the local Generic Transport so that they can open a Generic Transport SAP and communicate with each other. The Independent Alarm ME also registers its title with the Directory ME so that the Dependent Alarm ME can locate its address. The Dependent Alarm ME uses the Directory ME's title translation services to obtain the Independent Alarm ME's address. See chapter 5 in this manual for more information.

Statistics Manager

The Dependent Alarm ME uses the services of the Statistics Manager to generate statistics such as the number of alarm messages generated from a specific system, the number of discarded alarms, and so on.

The Dependent Alarm ME opens a statistics SAP, collects the statistics, and provides a procedure which is used by the Statistics Manager to report statistics. See the CDCNET Systems Programmer's Reference Manual, Volume 1, for more information on the Statistics Manager.

Services Provided

The external services provided by the Dependent Alarm ME to its users are the same as those provided by the Dependent Log ME. See Services Provided in chapter 8 of this manual for a description of these services.

Echo ME 10

Echo ME 10

This chapter discusses:

- An overview of Echo ME
- The services required by Echo ME
- The services provided by Echo ME
- Constants and common types used in Echo ME service requests

Overview

The Echo Management Entity (ME) is a software component that exists at a dedicated Internet SAP (3B SAP) in all DIs. See chapter 16 in this manual for more information on Internet layer.

It is used primarily by Internet users to do the following:

- Verify that a particular system in the catenet is operational and that a data path to it exists.
- Send a message to a particular system and have it returned.

All interfaces to Echo ME are through the Internet layer. Service requests from its users are sent through the Internet datagram request procedure with an echo operation field prefixed to the text portion of the message.

When the Echo ME receives an Internet Protocol Data Unit (3B PDU) from the Internet layer, it does the following:

- Verifies that the packet type is echo and that the value in the echo operation field is set to ECHO_REQUEST.
- Changes the value in the operation field to ECHO_REPLY.
- Sends the PDU back to the message source.

If an echo request message is checksummed, the echo reply packet is also checksummed. If a message transfer is successful and Echo ME is able to reply, the user receives the same datagram that it sent, with the echo operation field toggled to ECHO_REPLY.

If there are any problems in making the echo request, the Echo ME forwards the PDU to the Error ME in an error message request. The software component that originated the echo packet is sent an Internet Error Report indicating the type of error that occurred. See chapter 11 in this manual for more information on Error ME.

Echo ME's services can be used in the following ways:

- To verify that a particular DI system exists and is operational, and that a data path to it can be established. This can be tested by sending an echo request packet to that system and then waiting for a reply.
- To indicate that if there was a garbled message in an echo reply packet, that the message was transferred incorrectly somewhere along the path as the Echo ME does not alter the messages that it sends back.
- To determine the number of DI systems on a network solution by broadcasting an echo request packet on that network solution and then counting the echo replies.

Figure 10-1 illustrates the Echo ME Functions. Figure 10-2 illustrates the functional relationship of Echo ME with the Internet layer and Error ME.

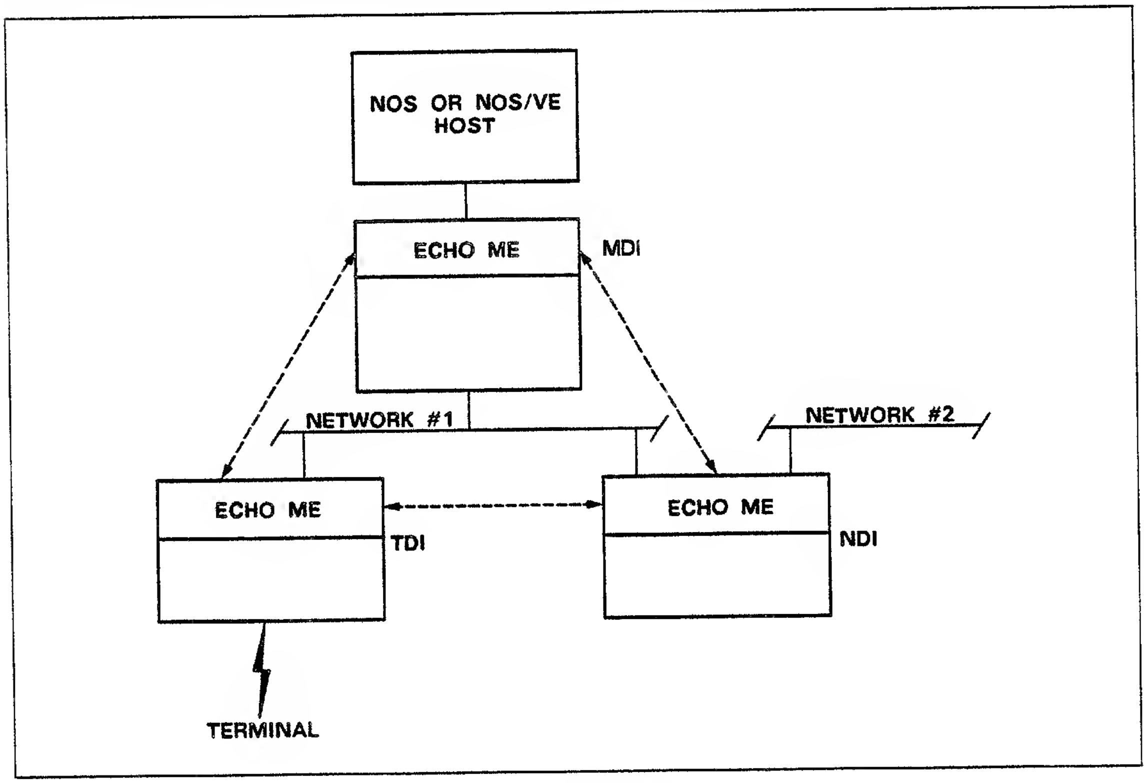


Figure 10-1. Echo ME Functions

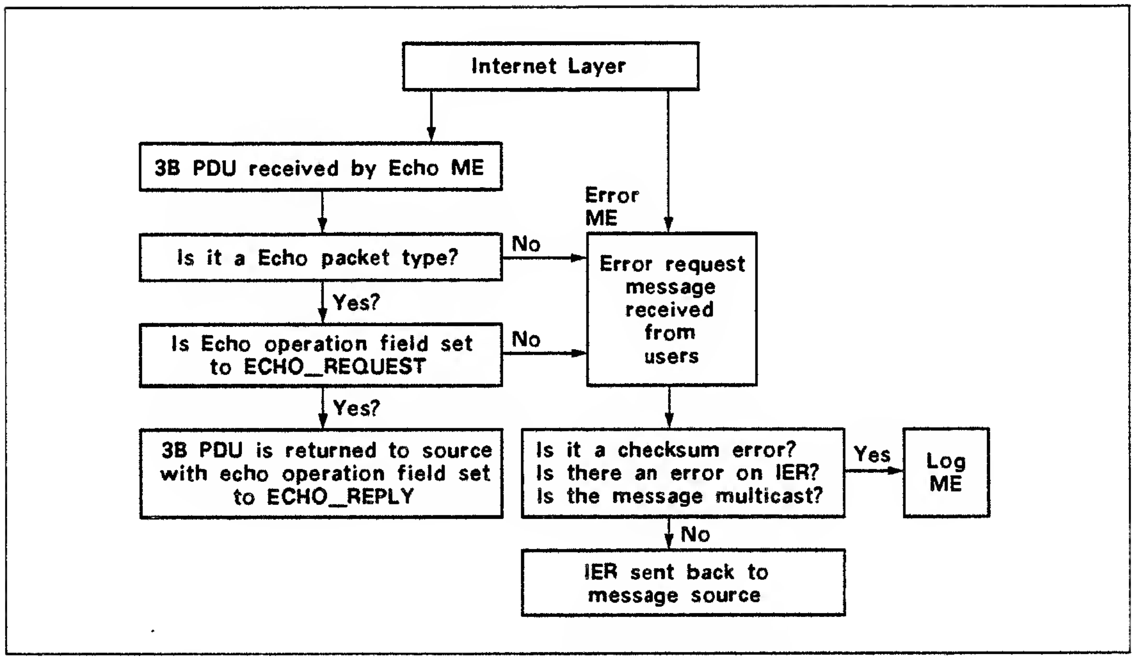


Figure 10-2. Echo, Error, and Internet Layer Relationship

Services Required

For the Echo ME to provide services to its own users it, in turn, depends on the services of other software components. Brief descriptions of each of these software components follow.

Internet Layer

The Echo ME uses the services of the Internet layer to deliver the echo packet and return it to its source. See chapter 16 in this manual for information on the Internet layer interfaces.

Error ME

Echo ME uses the services of Error ME when there are errors in an echo request packet it has received.

Services Provided

The Echo ME services are accessed through the Internet Layer. The user sends an ECHO_REQUEST packet to the dedicated Echo ME SAP. See chapter 16 in this manual for details.

Constants and Common Types

This section lists the constants and common types used in the Echo ME service requests. These data types are defined in the Internet layer's common decks. See Constants and Common Types in chapter 16 for complete information.

Constants

```
echo_me_sapid = 2

ier_not_echo_packet = 1001(16)

ier_not_echo_req = 1002(16)

xerox_echo_packet = 2
```

Common Types

```
echo_operation = (echo_null, echo_request, echo_reply)
```

Error ME 11

Error ME 11

This chapter discusses:

- An overview of the Error Management Entity (ME)
- The services required by the Error ME
- The services provided by the Error ME
- Constants and common types used in Error ME service requests

Overview

The Error ME is a software component that exists in all DIs and NOS/VE hosts. It is provided to help its users report errors that have been detected. The Error ME services are primarily used by the Internet layer and its users.

Error ME creates Internet Error Reports (IERs) and sends them to the software component that initiated the message in error. In some cases, Error ME logs the IERs to a log file. The IER is an Internet Protocol Data Unit (3B PDU). The standard 3B header is followed by error data (ERROR_ME_DATA) and the first 42 bytes of the 3B PDU which caused the error.

The Error ME receives 3B PDUs that are in error from its users (primarily the Internet layer), through an intertask message interface. The task ID of the Error ME is globally known. The 3B PDUs are sent with an error number, error parameter, and the 3B SAP ID of the software component that detected the error. When Error ME receives a 3B PDU, it builds an IER and sends the IER back to the originator of the message. The address of the originator is contained in the 3B PDU that was sent to the Error ME. For more information on the format of the 3B PDU, see the CDCNET Systems Programmer's Reference Manual, Volume 3.

Figure 11-1 shows Error ME in a NOS environment. Figure 11-2 shows Error ME in a NOS/VE environment.

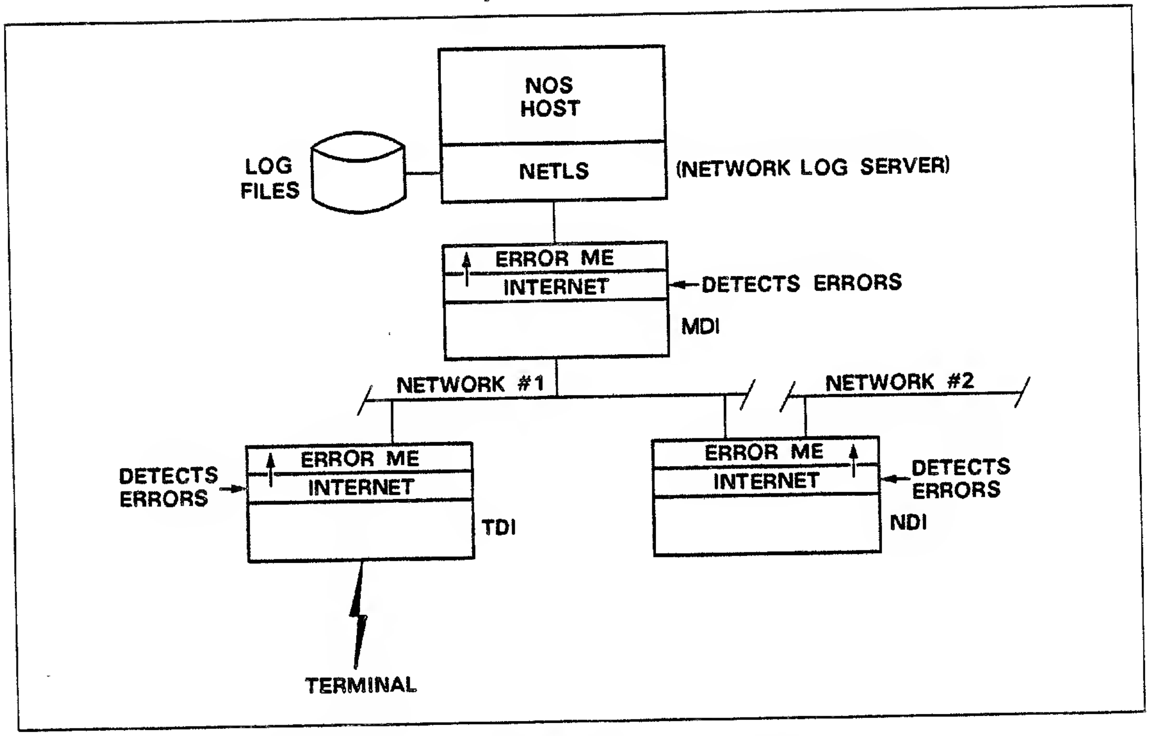


Figure 11-1. Error ME in a NOS Environment

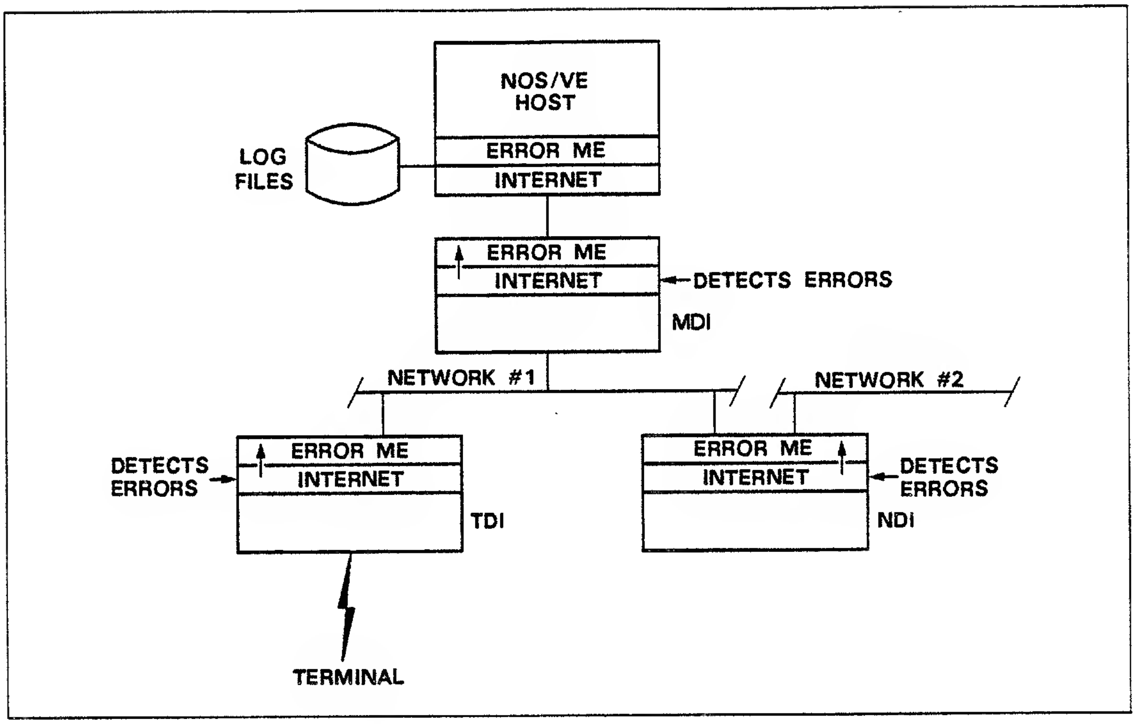


Figure 11-2. Error ME in a NOS/VE Environment

Following are the instances in which the IER is written to a log file.

Checksum errors	When a checksum error occurs, the IERs are logged because the integrity of the Internet header is in doubt and the message source address in the header may be incorrect.
Multicast messages	When an error occurs on a multicast message, the IERs are logged because one multicast may result in several data indications.
IER errors	When the source of the IER is Error ME itself, errors are logged instead of being sent to the message source.

Services Required

For the Error ME to provide services to its own users it, in turn, depends on the services of the other software components. Brief descriptions of the services of each of these software components follow.

Internet Layer

The Error ME uses the services of the Internet layer to send the IERs back to the message source. See chapter 16 in this manual for more information on the Internet layer.

Log ME

The services of the Log ME are used to log certain IERs that cannot be returned to the message source. See chapter 8 in this manual for more information on the Log ME.

Services Provided

This section describes the external services provided by the Error ME to its users. It describes the service requests exchanged between the Error ME and its users.

As mentioned earlier, Error ME services are used by the Internet layer and its users. Service requests to Error ME from these users are sent through an intertask message (ITM) called MSG_TO_ERRORME. The task identifier of the Error ME is globally known. Following is the Error ME task identifier:

```
errorme_taskid: (XREF) task_ptr;
```

The Error ME receives the MSG_TO_ERRORME ITM with the error code, error parameter, and the 3B PDU in error. The Error ME then generates an IER and sends it back to the originator of the message, using Internet layer's services. The address of the software component that generated the bad PDU is determined from the 3B PDU header.

The IER contains at least 42 bytes of the 3B PDU that caused the error. If the error is a checksum error, an IER error, or the message is multicast, the IER is logged into a log file.

Following is the description of the ITM sent by Error ME users. See chapter 16 in this manual for details on Internet layer interfaces.

MSG_TO_ERRORME

This ITM allows users to request Error ME to generate IERs.

Comdeck **B3DMEME**

Format **MSG_TO_ERRORME**
workcode: 0 .. 0ffff(16)
multicast: boolean
error_number: 0 .. 0ffff(16)
error_parameter: 0 .. 0ffff(16)
pdu_3b: buf_ptr

Input The user sets up the following fields as input in the MSG_TO_ERRORME record:

workcode: 0 .. 0ffff(16)

The ITM workcode. The following is the only valid workcode currently defined:

ERR_ME_INTERNET_ERROR

Any other value specified in this field will result in an error.

multicast: boolean

This field contains the following values:

TRUE	This is a multicast message.
FALSE	This is not a multicast message.

error_number: 0 .. 0ffff(16)

This field identifies the type of error. These numbers are assigned by the Internet layer. A complete list of currently defined error numbers is given under Constants and Common Types in chapter 16.

error_parameter: 0 .. 0ffff (16)

This field is used to further qualify an error number. It is used for certain kinds of errors. If a specific error parameter does not exist, then this field should be set to zero.

pdu_3b: buf_ptr

The pointer to the 3B PDU that was in error.

Constants and Common Types

This section lists the constants and common types used in the Error ME service requests. These data types are defined in the Internet layer's common decks. See Constants and Common Types in chapter 16 for complete information.

Constants

```
error_me_sapid = 3

ier3bpdu_size = 42

xerox_error_packet = 3
```

Common Types

```
error_me_data = record
  error_number: internet_error_codes,
  error_parameter: 0 .. 0ffff(16),
recend
```

Clock ME 12

Clock ME 12

This chapter gives an overview of the Clock ME.

Each CDCNET DI has a real-time clock and calendar on its main processor board. The Clock ME is responsible for managing and synchronizing the clocks in the DIs with the network. Clocks in the DI need to be synchronized because dates and time are important factors in the operation of the network.

Dates and time are reported by each CDCNET system in command responses, in log messages, and in alarms. In addition, some software components use a time stamp field in their protocol data units (PDU). For the time stamps to be meaningful across systems, the clocks have to be synchronized to the same time base.

There is one and only one master clock in every catenet. This master clock is defined through configuration commands and is used to synchronize all the clocks in the catenet.

The Clock ME is divided into the following two software components:

- Independent Clock ME
- Dependent Clock ME

Neither the Independent nor the Dependent Clock ME has external interfaces that a user, such as a gateway program, can use. Therefore, detailed information on this software component will not be provided in this manual. See the CDCNET Operations Manual for more information on the network commands used to manage the DI clocks.

Independent Clock ME

The Independent Clock ME contains the master clock and is responsible for synchronizing the clocks in all the systems within the catenet. The Independent Clock ME can reside in any DI or NOS/VE host. In a NOS environment, a site can identify, through configuration commands, which DI will be responsible for synchronizing the network's clocks. In a NOS/VE environment, the START_UP file identifies the host that is responsible for synchronizing the network's clocks. (See NOS/VE Interface Manual for more information on the START_UP file.) The Independent Clock ME responds to requests from the Dependent Clock MEs by providing the date and time from the master clock.

Dependent Clock ME

The Dependent Clock ME resides in every DI and is responsible for setting its system clock with the date and time provided by the Independent Clock ME.

When a DI is reset or when a network operator issues an appropriate CDCNET network command to ensure that all clocks in the network are synchronized, the Dependent Clock ME always requests the current date and time from the Independent Clock ME. The Independent Clock ME responds by transmitting an updated time to the Dependent Clock ME and the Dependent Clock ME adjusts the DI's real-time clock and calendar accordingly.

Figures 12-1 and 12-2 show the interrelationship between Independent and Dependent Clock MEs in NOS and NOS/VE environments respectively.

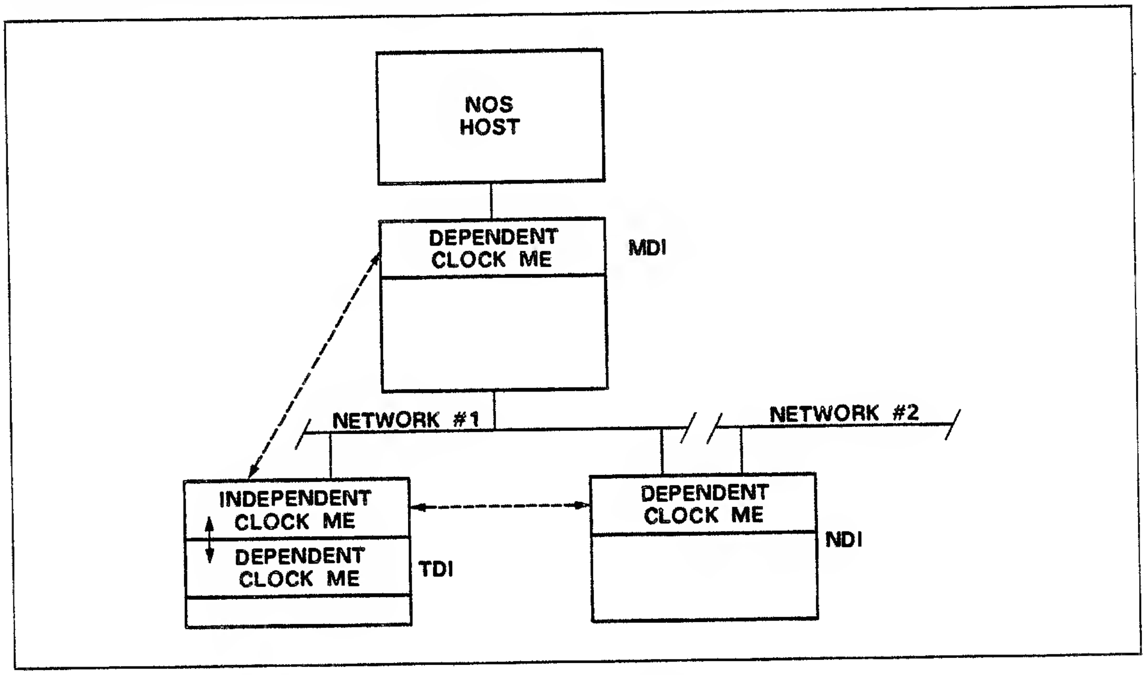


Figure 12-1. Clock ME in a NOS Environment

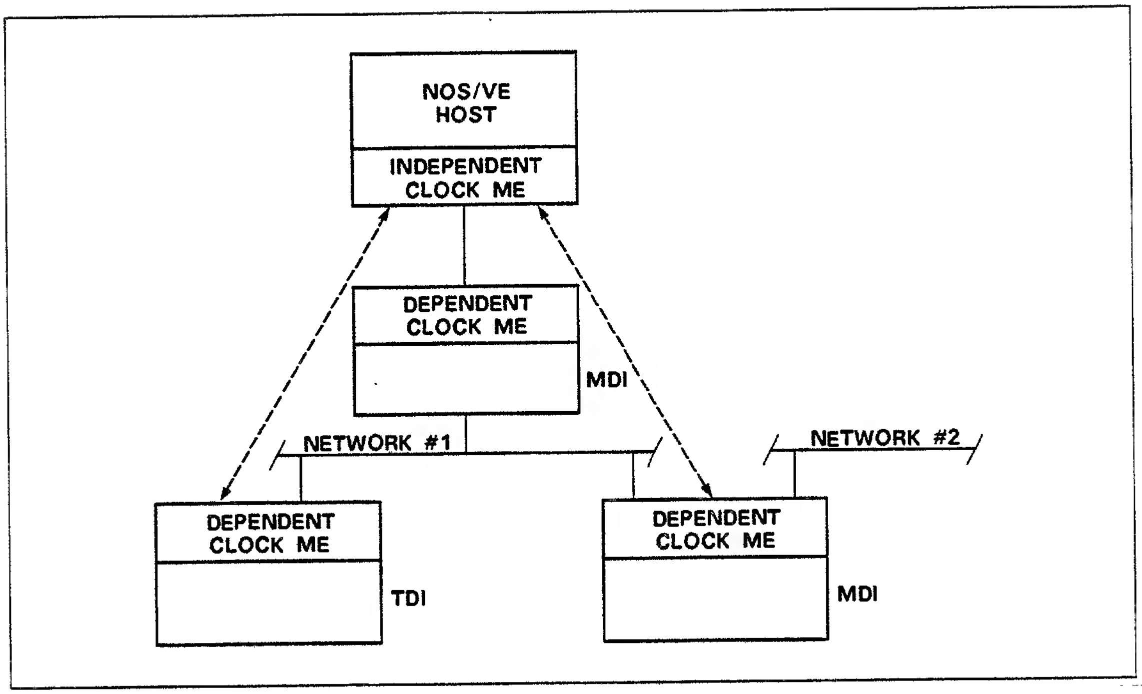


Figure 12-2. Clock ME in a NOS/VE Environment

102684212 1 OF 2

(Continued)

OSI Model

LAYER	NAME	
7	Application Layer	
6	Presentation Layer	
5	Session Layer	
4	Transport Layer	
3	Network Layer	
2	Data Link Layer	
1	Physical Layer	

CDNA

LAYER	NAME	
7 & 6	CDNA Higher Layers	
5	Session Layer	
4	Transport Layer • Generic Transport • Xerox Transport	
3B	Internet Layer	
3A	Intranet Layer	
2	Data Link Layer • MCI SSR • ESCI SSR • HDLC SSR	
1	Physical Layer	

Note: Shaded areas indicate layers described in this manual.

OSI and CDNA Layers

Network Layer Interfaces

Part III describes the CDCNET layer software. The layer software provides the functions needed to support communication between computer systems, terminals, applications, and end users connected to CDCNET. This software is based on the Control Data Network Architecture (CDNA). The following layers are described in this manual and shown in the figure on the other side of this divider.

Session layer (Layer 5)	This layer enables systems in the network to maintain an orderly dialogue.
Transport layer (Layer 4)	This layer matches the OSI model layer 4. Together, the following two software components assure end-to-end, transparent data transfer between two users. • Generic Transport layer • Xerox Transport layer
Network layer (Layer 3)	CDNA divides the OSI's layer 3 into two sublayers: • Internet layer (3B) • Intranet layer (3A) The Internet layer is responsible for relaying data from its source to its destination regardless of the number of network solutions between them. The Intranet layer is responsible for routing and relaying information within a specific network.
Data Link layer (Layer 2)	This layer ensures error-free transfer of data. It is identical to OSI model layer 2, and is composed of the following three software components: • Mainframe Channel Interface (MCI) Stream Service Routines (SSR) • Ethernet Channel Interface (ESCI) SSR • High-Speed Data Link Channel (HDLC) SSR

(Continued on other side)

Session Layer 13

Session Layer 13

This chapter discusses:

- An overview of the session layer (layer 5).
- The services required by the Session layer.
- The services provided by the Session layer.
- Constants and common types used in the Session layer's service requests.

Overview

The Session layer is a higher-layer software component that provides the means for applications such as gateway programs and terminal interface programs (TIPs) to organize and synchronize their dialogue and manage their data exchange. This ensures that users on one system are able to communicate with users on another system. The Session Layer does the following:

- Supports synchronization of connections. This allows its users to control the discarding of either transmit or receive data or both.
- Supports flow control which is implemented by Xerox Transport. The Session layer directly maps its own flow control service to Generic Transport's flow control service. See chapters 14 and 15 in this manual for more information.

A session connection is a virtual communication channel that temporarily connects two users so that they can exchange data. To implement the transfer of data between two users, the session connection is mapped into and uses a transport connection. The session connection is created when the user opens a Session layer service access point (SAP) and requests a connection. The user that initiated the connection designates the destination through a transport address.

The types of services provided by the Session layer are:

- Layer management services.
- Connection management services.
- User layer management services.
- User connection management services.

Layer management services and connection management services (Session layer services) are available to applications that reside in the higher layers. These applications may include gateway programs and TIPs. The Session layer could reside in a Mainframe Device Interface (MDI), a Mainframe Terminal Interface (MTI), a Network Device Interface (NDI), or a Terminal Device Interface (TDI).

Session layer services are provided by two procedure calls. One procedure call is used for layer management services and the other for connection management services. Each of these procedures is called with one parameter. The parameter is a variant record which contains a workcode and parameters which are dependent on this workcode. The workcode represents the service being requested or the response to an indication sent out earlier by the Session layer. The parameters can be either input or output parameters. A return status code parameter conveys the success or failure of the request being processed.

The layer management procedure is an externally referenced (XREF) procedure. The connection management procedure's entry point is returned to the user after the Session layer SAP is opened. See Services Provided, later in this chapter, for details.

User layer management services and user connection management services (user-provided services) are also provided by two procedure calls that are called by the Session layer to send indications or confirmations.

Each of these procedures is also called with one parameter -- a variant record which contains a workcode and parameters that are workcode-dependent. The workcode represents the service indications received from the Session layer or confirmations of processed requests. The parameters can be either input or output parameters.

The user initiates communication with the Session layer. This is done by calling the layer management procedure with an SL_OPEN_SAP request, and establishing a connection. The Session layer returns the entry point address of the connection management procedure which the user calls to regulate the dialogue with its peer. Figure 13-1 illustrates the functional relationship between the Session layer and its user.

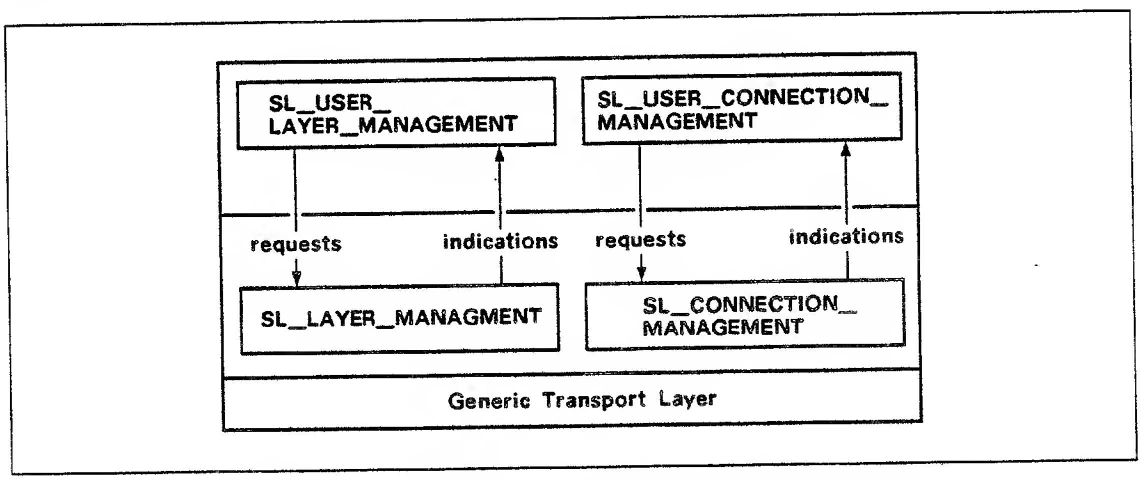


Figure 13-1. Session Layer Functional Relationship

Services Required

For the Session layer to provide services to its own users it, in turn, depends on the services of other software components. Brief descriptions of the services of each of these software components follow.

Directory ME

The Directory ME's translation, and wait service is used by the Session layer to translate Generic Transports title to obtain the Generic open SAP entry point address. See chapter 5 in this manual for details.

Transport Interface

Multiple Generic Transport SAPs are opened and used by the Session layer to provide reliable and sequenced data transfer for all its users. One Generic Transport SAP is opened for each Session layer SAP opened by a user. Chapter 14 in this manual provides the details.

Dependent Log ME

The Dependent Log ME (Log Support Application) is used to log abnormal Session layer events. See chapters 8 and 9 in this manual for details.

Statistics Manager

The CDCNET Statistics Manager collects and reports statistics. The following Session layer statistics are reported:

- The number of PDUs and characters received from the Transport layer.
- The number of PDUs and characters received from the Session layer user.
- The number of PDUs and characters transmitted by the peer or remote entity for each SAP.

See the CDCNET Systems Programmer's Reference Manual, Volume 1, for more information on statistics management.

Executive Interface

The Session layer uses well-defined interfaces to the Executive. These interfaçes are described in the CDCNET Systems Programmer's Reference Manual, Volume 1.

Services Provided

This section describes the external services provided by the Session layer to its users. It describes the major data structures used and the service requests exchanged between the Session layer and its users.

Layer Management Services

Layer management services perform the preliminary tasks of opening and closing a Session layer SAP and establishing a connection.

SL_LAYER_MANAGEMENT Procedure

Users interface with the Session layer through direct calls to the SL_LAYER_MANAGEMENT procedure. The calls are made with a single parameter which points to a variant record that contains both a workcode and parameters which are dependent on this workcode. Session layer users include the common deck SLXSESS in their calling modules. The common deck SLXSESS contains the following procedure declaration:

```
PROCEDURE [XREF] sl_layer_management (
    VAR request: sl_layer_mgmt_request );
```

The common deck SLXSESS contains a call to common deck SLDLCMD. SLDLCMD contains the layer management record, the connection management procedure, the user's layer management and connection management procedures, and all the type definitions.

Layer Management Request Record

The layer management request is the main data structure used by Session layer users to open or close a Session layer SAP and to establish a connection.

This data structure is a variant record defined with fields to support both input and output parameters. The workcode, service SAP identifier, and status fields are common and are used in all layer management requests. A value specified in the workcode field determines the other fields or service requests that are to be used in the layer management request.

Following is a CYBIL description of the layer management request record.

```
TYPE
sl_layer_mgmt_request = record
  workcode: sl_layer_mgmt_codes,
  service_sapid: gt_sap,
  status: sl_return_status,
  case sl_layer_mgmt_codes of

    = sl_open_sap =
        open_sap@: record
            user_sapid: ^cell,
            user_layer_mgmt_if: sl_user_layer_mgmt_call,
             user_connection_mgmt_if: sl_user_connection_mgmt_call,
             sl_connection_mgmt_if: sl_connection_management_call,
      recend,

    = sl_close_sap =        {No parameters
      ,

    = sl_call_request =
        call_request@: record
            user_cepid: ^cell,
            service_cepid: ^cell,
            destination_address: gt_sap,
            credit_window: 1 .. sl_max_credit,
            call_block: buf_ptr,
      recend,
 casend,
recend;
```

Following are the common fields defined in the static portion of the layer management request record:

workcode: sl_layer_mgmt_codes

This field must contain one of the following values:

SL_OPEN_SAP

SL_CLOSE_SAP

SL_CALL_REQUEST

The user must supply one of these values to specify a service request when making a layer management request. Details on these service requests are given later in this section.

service_sapid: gt_sap

This is the Session layer's SAP identifier. This parameter is returned to the user by the Session layer as an output parameter in the SL_OPEN_SAP request. The user specifies this identifier when making a CLOSE_SAP or a CALL_REQUEST request.

status: sl_return_status

This is the status indication of a processed request returned by the Session layer. Return codes are listed with each service request, later in this chapter. A complete list of status messages with explanations is also given in Constants and Common Types, later in this chapter.

Layer Management Service Requests

The following pages describe the various service requests offered to a Session layer user. The user requests a service by specifying an appropriate value in the workcode field of the layer management request record. The common fields described earlier are listed again in the following descriptions of the service requests. The explanation of each common field is not repeated unless there is some unique information for that field in a particular service request.

SL_OPEN_SAP

This request does the following:

- Allows the user to identify itself to the Session layer and to get the address of the Session layer's connection management procedure.
- Gets the addresses of the user's layer management and connection management procedures. These procedures are used by the Session layer to send indications and confirmations to the user's requests.

A value of SL_OPEN_SAP in the workcode field of the layer management request record initiates this request.

Input The user sets up the following fields as input in the layer management request record.

workcode: sl_layer_mgmt_codes

This field must contain the following value:

SL_OPEN_SAP

user_sapid: ^cell

The user's SAP identifier.

user_layer_mgmt_if: sl_user_layer_mgmt_call

This is the address of the user's layer management interface. It is used by the Session layer to send indications and confirmations to layer management requests.

user_connection_mgmt_if: sl_user_connection_mgmt_call

This is the address of the user's connection management interface. It is used by the Session layer to send indications and confirmations to connection management requests.

Output The Session layer returns the following fields as output in the layer management request record:

service_sapid: gt_sap

sl_connection_mgmt_if: sl_connection_management_call

This is the address of the Session layer's connection management interface. The user sends all connection management requests and responses to it.

status: sl_return_status

This is the status indication for the processed request. Following are the status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
sl_no_errors
sl_open_sap_error
sl_unknown_request_workcode
sl_unable_to_initialize
```

SL_CLOSE_SAP

This request allows the user to close a currently open Session layer SAP. All established connections associated with this SAP are terminated without notifying the user.

A value of SL_CLOSE_SAP in the workcode field of the layer management request record initiates this request.

Input The user sets up the following fields as input in the layer management request record:

workcode: sl_layer_mgmt_codes

This field must contain the following value:

SL_CLOSE_SAP

service_sapid: gt_sap

The Session layer's SAP identifier that is being closed.

Output The user sets up the following field as output in the layer management request record:

status: sl_return_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
sl_no_errors
sl_invalid_service_sapid
```

SL_CALL_REQUEST

This request allows the Session layer user to:

- Establish a connection with a peer Session layer user.
- Specify or negotiate the manner and content of the established connection.

A value of SL_CALL_REQUEST in the workcode field of the layer management request record initiates this request.

Input The user sets up the following fields as input in the layer management request record:

workcode: sl_layer_mgmt_codes

This field must contain the following value:

SL_CALL_REQUEST

service_sapid: gt_sap

user_cepid: ^cell

The user's CEPID.

destination_address: gt_sap

The network address of the peer.

credit_window: 1.. sl_max_credit

The number of received Generic Transport credits. The allowed range is 1 through 8. See chapter 14 in this manual for details.

call_block: buf_ptr

A pointer to a buffer chain. This buffer chain contains accounting, validation, and user data. The first two fields of the call block are length fields for the accounting and validation data records. Each length field is 16 bits long. The maximum user data size is 512 bytes.

Output The user sets up the following fields as output in the layer management request record.

service_cepid: ^cell

status: sl_return_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
sl_no_errors
sl_invalid_service_sapid
sl_invalid_data_size
sl_transport_rejected_connect
```

CAUTION

The user should not send a CALL_BLOCK buffer that has multiple users to the Session layer.

Connection Management Services

Connection management services manage the activity on connections. Connection management services include:

- Monitoring the exchange of both normal and expedited data.
- Synchronizing and terminating Session layer connections.
- Handling flow control.

SL_CONNECTION_MANAGEMENT Procedure

Session layer connection management requests are made by calling the SL_CONNECTION_MANAGEMENT procedure. The address of this procedure is passed to the user as an output parameter in an SL_OPEN_SAP request. The connection management request passed in the procedural call is defined below:

```
TYPE
   sl_connection_management_call = ^procedure (
      VAR request: sl_connection_mgmt_request);
```

Connection Management Request Record

The connection management request record is the data structure used by Session layer users to manage their data exchange. This data structure is a variant record with fields to support both input and output parameters. The workcode, service CEPID, and status fields are common and are used in all connection management requests. A value specified in the workcode field determines other fields or service requests that are to be used in the connection management request.

Following is a CYBIL description of the connection management request record.

```
TYPE
sl_connection_mgmt_request = record
  workcode: sl_connection_mgmt_codes,
  service_cepid: ^cell,
  status: sl_return_status,
  case sl_connection_mgmt_codes of

    = sl_call_response =
      call_response@: record
        credit_window: 1 .. sl_max_credit,
        call_block: buf_ptr,
    recend,

    = sl_clear_request =
      clear_request@: record
        clear_block: buf_ptr,
        session_statistics: buf_ptr,
    recend,

    = sl_data_request =
      data_request@: record
        q_bit: boolean,
        m_bit: boolean,
        user_data: buf_ptr,
    recend,

    = sl_interrupt_request =
      interrupt_request@: record
        data: buf_ptr,
    recend,

    = sl_synch_request =
      synch_request@: record
        send_receive_indicator: discard_indicator_type,
        data: buf_ptr,
    recend,

    = sl_synch_response =               {No parameters
      ,

    = sl_flow_control_request =
      flow_control_request@: record
        status: sl_flow_control_status,
    recend,
  casend,
recend;
```

Following are the common fields defined in the static portion of the connection management request record:

workcode: sl_connection_mgmt_codes

This field must contain one of the following values:

SL_CALL_RESPONSE

SL_CLEAR_REQUEST

SL_DATA_REQUEST

SL_INTERRUPT_REQUEST

SL_SYNCH_REQUEST

SL_SYNCH_RESPONSE

SL_FLOW_CONTROL_REQUEST

The user must supply one of these values to specify a service request when making a connection management request. Details on these service requests are given later in this section.

service_cepid: ^cell

The Session layer's CEPID.

status: sl_return_status

This is the status indication of a processed request returned by the Session layer. Return codes are listed with each service request, later in this chapter. A complete list of status messages with explanations is also given in Constants and Common Types, later in this chapter.

Connection Management Service Requests

The following pages describe the various service requests offered to a Session layer user. The user requests a service by specifying an appropriate value in the workcode field of the connection management request record. The common fields described earlier are listed again in the following descriptions of the service requests. The explanation to each common field is not repeated unless there is some unique information for that field in a particular service request.

SL_CALL_RESPONSE

This request allows the user to notify the Session layer that it accepts the connection request sent out by a peer entity.

A value of SL_CALL_RESPONSE in the workcode field of the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: sl_connection_mgmt_codes

This field must contain the following value:

SL_CALL_RESPONSE

service_cepid: ^cell

credit_window: 1 .. sl_max_credit

The number of received Generic Transport credits. The allowed range is 1 through 8. See chapter 14 in this manual for explanations.

call_block: buf_ptr

A pointer to a buffer chain. This buffer chain contains accounting, validation, and user data. The first two fields of the call block are length fields for the accounting and validation data records. Each length field is 16 bits long. The maximum user data size is 512 bytes.

Output The Session layer returns the following field as output in the connection management request record:

status: sl_return_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
sl_invalid_data_size
sl_no_errors
sl_invalid_for_state
sl_invalid_service_cepid
```

SL_CLEAR_REQUEST

This request enables the user to do one of the following:

- Refuse a connection that is being initiated by a peer.
- Terminate a previously established Session layer connection.

The Session layer returns to its user the statistics that it collected for that connection. Statistics include:

- Connect time.
- Number of characters sent and received.
- Number of blocks sent and received.

A value of SL_CLEAR_REQUEST in the workcode field of the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: sl_connection_mgmt_codes

This field must contain the following value:

SL_CLEAR_REQUEST

service_cepid: ^cell

clear_block: buf_ptr

A pointer to a buffer chain. This buffer chain contains 0 through 512 bytes of user-dependent clearing information.

Output The Session layer returns the following fields as output in the connection management request record:

session_statistics: buf_ptr

A pointer to a buffer chain. This buffer chain contains statistics that the Session layer collected for that particular connection. Statistics include connect time, number of characters sent and received, and number of blocks sent and received.

status: sl_return_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
sl_invalid_data_size
sl_no_errors
sl_invalid_for_state
sl_invalid_service_cepid
```

Remarks

A 30-second timer is set by the Session layer when it receives an SL_CLEAR_REQUEST. SL_CLEAR_REQUEST is sent as normal data to ensure that the previous data sent has reached the peer. The peer Session layer terminates the connection using the peer Generic Transport's disconnect service. Data that is being cleared can be potentially lost if the Session layer's timer expires before it receives a disconnect indication from Generic Transport. If a GT_DISCONNECT_INDICATION is not received from Generic Transport within 30 seconds, the Session layer issues a GT_DISCONNECT_REQUEST to Generic Transport.

SL_DATA_REQUEST

This request allows the Session layer user to exchange data with its peer on an established connection.

A value of SL_DATA_REQUEST in the workcode field of the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: sl_connection_mgmt_codes

This field must contain the following value:

SL_DATA_REQUEST

service_cepid: ^cell

q_bit: boolean

The data qualifier bit in the call request packet. This field contains one of the following values:

TRUE	The data packet contains control information about the data stream that is to be interpreted by a higher-level protocol.
FALSE	The data packet contains unqualified information.

m_bit: boolean

The more data bit in the call request packet. This field contains one of the following values:

TRUE	More data follows this data packet.
FALSE	This data packet completes a packet sequence.

user_data: buf_ptr

The data to be transmitted.

Output The Session layer returns the following field as output in the connection management request record:

status: sl_return_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
sl_no_errors
sl_invalid_for_state
sl_invalid_service_cepid
```

SL_INTERRUPT_REQUEST

This request allows the Session layer users to bypass normal data on a connection. There is no confirmation of delivery returned on this request; thus, more than one interrupt may be outstanding at any time. The interrupt data is limited to 14 bytes because Generic Transport has limitations on expedited data.

A value of SL_INTERRUPT_REQUEST in the Connection Management Request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: sl_connection_mgmt_codes

This field must contain the following value:

SL_INTERRUPT_REQUEST

service_cepid: ^cell

data: buf_ptr

The expedited data to be transmitted. The value is limited to a range of 1 through 14 bytes.

Output The Session layer returns the following field as output in the connection management request record:

status: sl_return_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
sl_invalid_data_size
sl_no_errors
sl_invalid_for_state
sl_invalid_service_cepid
```

SL_SYNCH_REQUEST

This request allows the Session layer user to synchronize a particular connection.

A value of SL_SYNCH_REQUEST in the workcode field of the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: sl_connection_mgmt_codes

This field must contain the following value:

SL_SYNCH_REQUEST

service_cepid: ^cell

send_receive_indicator: discard_indicator_type

This field indicates the direction in which data is to be discarded. Send and receive are directions relative to the user initiating the synchronization request. This parameter contains one of the following values:

DISCARD_RECEIVE	Discard the data received.
DISCARD_SEND	Discard the data sent.
DISCARD_SEND_ RECEIVE	Discard data sent and received.

data: buf_ptr

The user-dependent synchronization data (reason code). The value is limited through a range of 1 through 14 bytes.

Output The Session layer returns the following field as output in the connection management request record:

status: sl_return_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
sl_invalid_data_size
sl_no_errors
sl_invalid_for_state
sl_invalid_service_cepid
```

Remarks The synchronization request always causes expedited data to be sent to the peer Session layer along with a synchronization indication to the peer user. Synchronization data is limited to 14 bytes because Generic Transport has limitations on expedited data. Only one synchronization request can be outstanding at any time. The user can, however, issue a data request while waiting for a confirmation to the synchronization request.

SL_SYNCH_RESPONSE

This request allows the user to inform the Session layer that it has processed a DISCARD_SEND_RECEIVE or DISCARD_RECEIVE synchronization request initiated by its peer. This request is in response to a synchronization indication with a DISCARD_SEND_RECEIVE or a DISCARD_RECEIVE option. When received by the peer, the request marks the end of the data to be discarded, thus completing the synchronization process.

A value of SL_SYNCH_RESPONSE in the workcode field of the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: sl_connection_mgmt_codes

This field must contain the following value:

SL_SYNCH_RESPONSE

service_cepid: ^cell

Output The Session layer returns the following field as output in the connection management request record:

status: sl_return_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
sl_no_errors
sl_invalid_for_state
sl_invalid_service_cepid
```

NOTE

This request should not be used in response to a sychronization indication with a DISCARD_SEND option.

SL_FLOW_CONTROL_REQUEST

This request allows the Session layer user to request flow control of data.

The Session layer directly maps this service to Generic Transport's flow control service, which is responsible for stopping and starting the flow of data. See chapter 14 in this manual for details.

A value of SL_FLOW_CONTROL_REQUEST in the workcode field of the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: sl_connection_mgmt_codes

This field must contain the following value:

SL_FLOW_CONTROL_REQUEST

service_cepid: ^cell

status: sl_flow_control_status

This field indicates if the Transport layer should start sending data or stop sending data. This field contains one of the following values:

SL_START_DATA

SL_STOP_DATA

Output The Session layer returns the following field as output in the connection management request record:

status: sl_return_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
sl_no_errors
sl_invalid_for_state
sl_invalid_service_cepid
```

User Layer Management Services

User layer management services are provided by the Session layer user. The Session layer uses these services to inform its user that a Session SAP is closing and to deliver call indications to the user.

SL_USER_LAYER_MGMT Procedure

The user layer management indications are sent by calling the SL_USER_LAYER_MGMT procedure. The address of this procedure is passed as an input parameter in an OPEN_SAP request. The layer management indication passed in the procedural call to the user is defined below:

```
TYPE
   sl_user_layer_mgmt_call = ^procedure (
      VAR indication: sl_user_layer_mgmt_ind);
```

User Layer Management Indication Record

The SL_USER_LAYER_MGMT_IND is the data structure that is used by the Session layer to indicate to its users that a Session SAP has closed, or to indicate that a peer user wishes to establish a connection.

This data structure is a variant record with fields to support both input and output parameters. The workcode and the service SAP identifier are common and are used in all user layer management indications. A value specified by the Session layer in the workcode determines the other fields or service indications that are to be used in the user layer management indication record.

Following is a CYBIL description of the user layer management indication record:

```
TYPE
sl_user_layer_mgmt_ind = record
  workcode: sl_user_layer_mgmt_codes,
  user_sapid: ^cell,
  case sl_user_layer_mgmt_codes of

    = sl_sap_closed =
      ,                                    {No parameters

    = sl_call_indication =
      call_indication@: record
        service_cepid: ^cell,
        user_cepid: ^cell,
        peer_service_sapid: gt_sap,
        call_block: buf_ptr,
    recend,
  casend,
recend;
```

Following are the common fields defined in the static portion of the user layer management indication record:

workcode: sl_user_layer_mgmt_codes

The workcode must contain one of the following values:

SL_SAP_CLOSED

SL_CALL_INDICATION

The Session layer must specify one of these values when sending an indication or confirmation to a layer management request. Details on these indications are given later in this section.

user_sapid: ^cell

The user's SAP identifier.

User Layer Management Requests

The following pages describe the various user indications sent to the Session layer. The Session layer requests a service by specifying an appropriate value in the workcode field of the user layer management indication record. The common fields described earlier are listed again in the following descriptions of the indications. The explanation to each common field is not repeated unless there is some unique information for that field in a particular indication.

SL_SAP_CLOSED

This indication allows the Session layer to inform the user that its services are no longer available and that all previously established connections have been terminated for this SAP.

A value of SL_SAP_CLOSED in the workcode field of the user layer management indication record initiates this request.

Input The Session layer sets up the following fields as input in the user layer management indication record:

workcode: sl_user_layer_mgmt_codes

This field must contain the following value:

SL_SAP_CLOSED

user_sapid: ^cell

Output None.

SL_CALL_INDICATION

This indication allows the Session layer to inform the user that a peer Session layer user wants to establish a connection.

A value of SL_CALL_INDICATION in the workcode field of the user layer management indication record initiates this request.

Input The Session layer sets up the following fields as input in the user layer management indication record:

workcode: sl_user_layer_mgmt_codes

This field must contain the following value:

SL_CALL_INDICATION

user_sapid: ^cell

service_cepid: ^cell

The Session layer's CEPID.

peer_service_sapid: gt_sap

The transport address of the user initiating the connection.

call_block: buf_ptr

A pointer to a buffer chain. This buffer chain contains 4 through 512 bytes of user-dependent call information.

Output The user returns the following fields as output in the user layer management indication record:

user_cepid: ^cell

The user's CEPID.

User Connection Management Services

The user connection management services are provided by the Session layer user. The Session layer uses these services to notify the user that:

- A connection request has been accepted or rejected.
- To indicate there is data for the user from its peer.
- To indicate synchronization and flow control requests for a connection.

SL_USER_CONNECTION_MGMT Procedure

The user connection management indications are sent by calling the SL_USER_CONNECTION_MGMT procedure. The address of this procedure is passed as an input parameter in an SL_OPEN_SAP request. The connection management indication passed in the procedural call to the user is defined below:

```
TYPE
  sl_user_connection_mgmt_call = ^procedure (
    VAR indication: sl_user_connection_mgmt_ind);
```

User Connection Management Indication Record

The SL_USER_CONNECTION_MGMT_IND is the data structure used by the Session layer to send indications related to data exchange.

This data structure is a variant record with fields to support both input and output parameters. The workcode and user CEPID parameters are common and are used in all user connection management indications. A value specified in the workcode field determines the other fields or indications that are to be used in the user connection management indication record.

The following is a CYBIL description of the user connection management indication record:

```
TYPE
sl_user_connection_mgmt_ind = record
  workcode: sl_user_connection_mgmt_codes,
  user_cepid: ^cell,
  case sl_user_connection_mgmt_codes of

    = sl_call_confirm =
      call_confirm@: record
        call_block: buf_ptr,
    recend,

    = sl_clear_indication =
      clear_indication@: record
        clear_origin: sl_clear_origin_types,
        clear_block: buf_ptr,
        session_statistics: buf_ptr,
    recend,

    = sl_data_indication =
      data_indication@: record
        q_bit: boolean,
        m_bit: boolean,
        user_data: buf_ptr,
    recend,

    = sl_interrupt_indication =
      interrupt_indication@: record
        data: buf_ptr,
    recend,

    = sl_synch_indication =
      synch_indication@: record
        send_receive_indicator: discard_indicator_type,
        data: buf_ptr,
    recend,

    = sl_synch_confirm =
      ,                                       {No parameters

    = sl_flow_control_indication =
      flow_control_indication@: record
        status: sl_flow_control_status,
    recend,

    = sl_expedited_flow_control_indic=
      expedited_flow_control_indic@: record
        status: sl_expedited_flow_control_stat,
      recend,

  casend,
recend;
```

Following are the common fields defined in the static portion of the user connection management indication record:

workcode: sl_user_connection_mgmt_codes

The workcode must contain one of the following values:

SL_CALL_CONFIRM

SL_CLEAR_INDICATION

SL_DATA_INDICATION

SL_INTERRUPT_INDICATION

SL_SYNCH_INDICATION

SL_SYNCH_CONFIRM

SL_FLOW_CONTROL_INDICATION

SL_EXPEDITED_FLOW_CONTROL_INDIC

The Session layer must specify one of these values when sending a connection management indication. Details on these values are given later in this section.

user_cepid: ^cell

The user's CEPID.

User Connection Management Indications

The following pages describe the various user indications offered to the Session layer. The Session layer requests a service by specifying an appropriate value in the workcode field of the user connection management indication record. The common fields described earlier are listed again in the following descriptions of the indications. The explanation to each common field is not repeated unless there is some unique information for that field in a particular indication.

SL_CALL_CONFIRM

This indication allows the Session layer to inform the user that a peer Session layer user accepts a connection initiated by the user.

A value of SL_CALL_CONFIRM in the workcode field of the user connection management indication record initiates this request.

Input The Session layer sets up the following fields as input in the user connection management indication record:

workcode: sl_user_connection_mgmt_codes

This field must contain the following value:

SL_CALL_CONFIRM

user_cepid: ^cell

call_block: buf_ptr

A pointer to a buffer chain. This record contains 4 through 512 bytes of user-dependent call-confirm information.

Output None.

SL_CLEAR_INDICATION

This indication allows the Session layer to inform the user that a previously established connection has been terminated or a connection request has been rejected by a peer user.

A value of SL_CLEAR_INDICATION in the workcode field of the user connection management indication record initiates this request.

Input The Session layer sets up the following fields as input in the user connection management indication record:

workcode: sl_user_connection_mgmt_codes

This field must contain the following value:

SL_CLEAR_INDICATION

user_cepid: ^cell

clear_origin: sl_clear_origin_types

This field identifies the software component responsible for the connection termination. It contains one of the following values:

SL_PEER_USER	A clear request was initiated by the peer Session layer user.
SL_TRANSPORT	A transport connection was terminated.
SL_SESSION_LAYER	The connection was terminated by the Session layer because of a protocol error.

clear_block: buf_ptr

A pointer to a buffer chain. This buffer chain contains information generated by the peer user.

session_statistics: buf_ptr

The statistics collected for a particular connection.

Output None.

Remarks The Session layer returns to its users statistics that it collected for that particular connection. Statistics include:

- Connect time.
- Number of characters sent and received.
- Number of blocks sent and received.

SL_DATA_INDICATION

This indication allows the Session layer to deliver to the user, on an established connection, the data sent by its peer.

A value of SL_DATA_INDICATION in the workcode field of the user connection management indication record initiates this request.

Input The Session layer sets up the following fields as input in the user connection management indication record:

workcode: sl_user_connection_mgmt_codes

This field must contain the following value:

SL_DATA_INDICATION

user_cepid: ^cell

q_bit: boolean

The data qualifier bit in the data request packet. This field contains one of the following values:

TRUE	The data packet contains control information about the data stream that is to be interpreted by a higher-level protocol.
FALSE	The data packet contains unqualified information.

m_bit: boolean

The more data bit in the data request packet. This field contains one of the following values:

TRUE	More data follows this data packet.
FALSE	This data packet completes a packet sequence.

user_data: buf_ptr

The received data that is to be delivered to the user.

Output None.

SL_INTERRUPT_INDICATION

This indication allows the Session layer to deliver to the user, on an established connection, interrupt data sent by its peer. The interrupt may have bypassed normal data on the connection as interrupts are not subject to flow control.

A value of SL_INTERRUPT_INDICATION in the workcode field of the user connection management indication record initiates this request.

Input The Session layer sets up the following fields as input in the user connection management indication record:

workcode: sl_user_connection_mgmt_codes

This field must contain the following value:

SL_INTERRUPT_INDICATION

user_cepid: ^cell

data: buf_ptr

Pointer to a buffer chain containing 1 through 14 bytes of expedited data.

Output None.

SL_SYNCH_INDICATION

This indication allows the Session layer to inform the user that a peer user initiated a request to synchronize a connection.

A value of SL_SYNCH_INDICATION in the workcode field of the user connection management indication record initiates this request.

Input The Session layer sets up the following fields as input in the user connection management indication record:

workcode: sl_user_connection_mgmt_codes

This field must contain the following value:

SL_SYNCH_INDICATION

user_cepid: ^cell

send_receive_indicator: discard_indicator_type

This field indicates the direction in which data is to be discarded. Send and receive are directions relative to the user initiating the synchronization request. This field contains one of the following values:

DISCARD_RECEIVE	Discard the data sent.
DISCARD_SEND	Discard the data received.
DISCARD_SEND_RECEIVE	Discard data sent and received.

data: buf_ptr

The user-dependent synchronization data (reason code). The allowed range is 1 through 14 bytes.

Output None.

NOTE

An SL_SYNCH_RESPONSE is required if the synchronization request is DISCARD_RECEIVE or DISCARD_SEND_RECEIVE.

SL_SYNCH_CONFIRM

This indication allows the Session layer to inform the user that the DISCARD_SEND_RECEIVE or DISCARD_RECEIVE synchronization request that it initiated has been completed.

A value of SL_SYNCH_CONFIRM in the workcode field of the user connection management indication record initiates this request.

Input The Session layer sets up the following fields as input in the user connection management indication record:

workcode: sl_user_connection_mgmt_codes

This field must contain the following value:

SL_SYNCH_CONFIRM

user_cepid: ^cell

Output None.

SL_FLOW_CONTROL_INDICATION

This indication allows the Session layer to inform the user that the Session layer has requested flow control on the data that is being sent.

A value of SL_FLOW_CONTROL_INDICATION in the workcode field of the user connection management indication record initiates this request.

Input The Session layer sets up the following fields as input in the user connection management indication record:

workcode: sl_user_connection_mgmt_codes

This field must contain the following value:

SL_FLOW_CONTROL_INDICATION

user_cepid: ^cell

status: sl_flow_control_status

This field indicates if the Transport layer should start or stop sending data. This field contains one of the following values:

SL_START_DATA

SL_STOP_DATA

Output None.

SL_EXPEDITED_FLOW_CONTROL_INDIC

This indication allows the Session layer to notify the user that there has been a change in the status of the underlying Transport layer's expedited data path.

A value of SL_EXPEDITED_FLOW_CONTROL_INDIC in the workcode field of the user connection management indication request record initiates this request.

Input The Session layer sets up the following fields as input in the user connection management indication record:

workcode: sl_expedited_flow_control_indic

This field must contain the following value:

SL_EXPEDITED_FLOW_CONTROL_INDIC

user_cepid: ^cell

status: sl_expedited_flow_control_stat

This field indicates the status of the Transport layer's expedited data path. This field contains one of the following values.

SL_EXPEDITED_START_DATA

SL_EXPEDITED_STOP_DATA

Output None.

Remarks The SL_INTERRUPT_REQUEST and the SL_SYNCH_REQUEST depend on Transport layer's expedited data service. Reacting to expedited data, flow control indication is highly recommended.

Constants and Common Types

This section lists the constants and common types used in the Session layer's service requests. It also lists and explains the status returned for the processed requests. These data types are defined in the common deck SLDLCMD.

Constants

```
sl_max_credit = 8
```

Common Types

```
discard_indicator_type = (
  discard_send_receive,
  discard_send,
  discard_receive)

sl_clear_origin_types = (
  sl_peer_user,
  sl_transport,
  sl_session_layer)

sl_connection_mgmt_codes = (
  sl_call_response,
  sl_clear_request,
  sl_data_request,
  sl_interrupt_request,
  sl_synch_request,
  sl_synch_response,
  sl_flow_control_request)

sl_flow_control_status = (
  sl_start_data,
  sl_stop_data)

sl_expedited_flow_control_stat = (
  sl_expedited_start_data,
  sl_expedited_stop_data)

sl_layer_mgmt_codes
  sl_open_sap,
  sl_close_sap,
  sl_call_request)

sl_return_status = (
  sl_no_errors,
  sl_open_sap_error,
  sl_invalid_service_sapid,
  sl_invalid_service_cepid,
  sl_invalid_for_state,
  sl_unknown_request_workcode,
  sl_transport_rejected_connect,
  sl_invalid_data_size,
  sl_no_di_resources,
  sl_unable_to_initialize
```

sl_no_errors	The request was processed without any errors.
sl_open_sap_error	The Session layer was unable to open a SAP.

sl_invalid_service_sapid	The Session layer at the destination does not recognize the initiating Session layer's network address.
sl_invalid_service_cepid	The Session layer at the destination does not recognize the initiating Session layer's CEPID.
sl_invalid_for_state	The request is invalid for the state of this connection.
sl_unknown_request_workcode	The received workcode was not SL_LAYER_MGMT_CODES or SL_CONNECTION_MGMT_CODES.
sl_transport_rejected_connect	Generic Transport rejected a CONNECT_REQUEST.
sl_invalid_data_size	The size of the data in a request exceeds the maximum specified for that request type.
sl_no_di_resources	There is insufficient memory for the execution of this request.
sl_unable_to_initialize	The Session layer's initialization procedure failed.

```
sl_user_connection_mgmt_codes = (
    sl_call_confirm,
    sl_clear_indication,
    sl_data_indication,
    sl_interrupt_indication,
    sl_synch_indication,
    sl_synch_confirm,
    sl_flow_control_indication,
    sl_accounting_indication)

sl_user_layer_mgmt_codes = (
    sl_sap_closed,
    sl_call_indication)
```

Transport Layer

CDCNET's Transport layer (layer 4) matches the OSI model. It is implemented in every system as two software components. Together, the following two software components assure end-to-end, transparent data transfer between two users:

- Generic Transport layer
- Xerox Transport layer

Generic Transport Layer

The Generic Transport layer is provided to support the ISO class 4 Transport. All CDCNET software requiring layer 4 services interface with the Generic Transport layer. The Generic Transport layer uses the Xerox Transport layer to help provide services to its users. Where there is no direct mapping between ISO services and Xerox Transport layer services, the Generic Transport layer provides the functions needed to support this mapping. Subsequent CDCNET releases may include support of standards other than the Xerox Transport layer. The use of the Generic Transport layer will reduce the need for Transport layer users to be concerned with the specifications of a particular transport layer standard.

Xerox Transport Layer

The Xerox Transport layer is provided so that CDCNET can support higher layer protocols like the Virtual Terminal Protocol (VTP). The Xerox Transport service is supported by the Xerox Sequenced Packet Protocol. Implementation of the Xerox Transport layer does not, however, preclude future applications from interfacing with the Xerox standard directly.

The following two chapters describe the Generic Transport layer and the Xerox Transport layer.

Generic Transport Layer 14

Generic Transport Layer 14

The Generic Transport layer portion of the Transport layer is a middle layer software component that does the following:

- Provides end-to-end, transparent data transfer between two users. It provides reliable data transfer from a source to a destination by establishing a logical connection between the source and the destination.
- Breaks up normal data messages into segments that can be transmitted individually.
- Reassembles these segments into the correct sequence upon reception.

Generic Transport layer services can be functionally divided into three groups:

- The layer management group
- The connection management group
- The indication group, consisting of two interfaces:
  - GENERIC_CONNECT_IF
  - GENERIC_DATA_IF

This chapter discusses:

- An overview of the Generic Transport Layer, which is part of the Transport layer.
- The services required by the Generic Transport layer.
- The services provided by the Generic Transport layer.
- Constants and common types used in the Generic Transport layer's service requests.

Overview

This section describes:

- User and service interfaces.
- The connection establishment process.
- Fragmentation and reassembly of data (done by the Generic Transport layer).
- Flow control (supported by the Generic Transport layer).
- Data transmission services (provided by the Generic Transport layer).
- Priorities assigned to connections.

User and Service Interfaces

As shown in figure 14-1, the Generic Transport layer can be viewed as having three sets of user interfaces and three sets of service interfaces.

User interface groups provide users with layer management and connection management services such as:

- Opening SAPs
- Establishing connections
- Transferring data on established connections
- Breaking a connection when it is no longer required
- Passing indications and confirmations

Service interfaces are between the Generic Transport layer and the Xerox Transport layer, and provide the means by which two transport layers communicate.

All the interfaces are invoked through procedure calls.

GENERIC_LAYER_MGMT	Used for opening or closing the Generic Transport layer SAPs and for establishing a connection.
GENERIC_CONNECTION_MGMT_CALL	Used for accepting connections, disengaging connections, transferring data, and imposing flow control.

The user supplies the addresses of two interfaces in an open SAP request which are used by the Generic Transport layer to send indications and confirmations.

USER_CONNECT_MGMT_IF	Used for sending connection-oriented indications and confirmations.
USER_LAYER_MGMT_IF	Used for sending all other indications.

When the Generic Transport layer is loaded in a system, it registers its title with the Directory ME. A user who wishes to communicate with the Generic Transport layer asks for a directory translation. The user receives the address of the GT_LAYER_MGMT_CALL procedure and sends a request to open a Generic Transport layer SAP. Once a SAP is opened, a connection is requested. A SAP may accept many incoming connections. Closing a connection does not affect the SAP associated with the connection, but closing the SAP aborts all the active connections on the SAP that was closed.

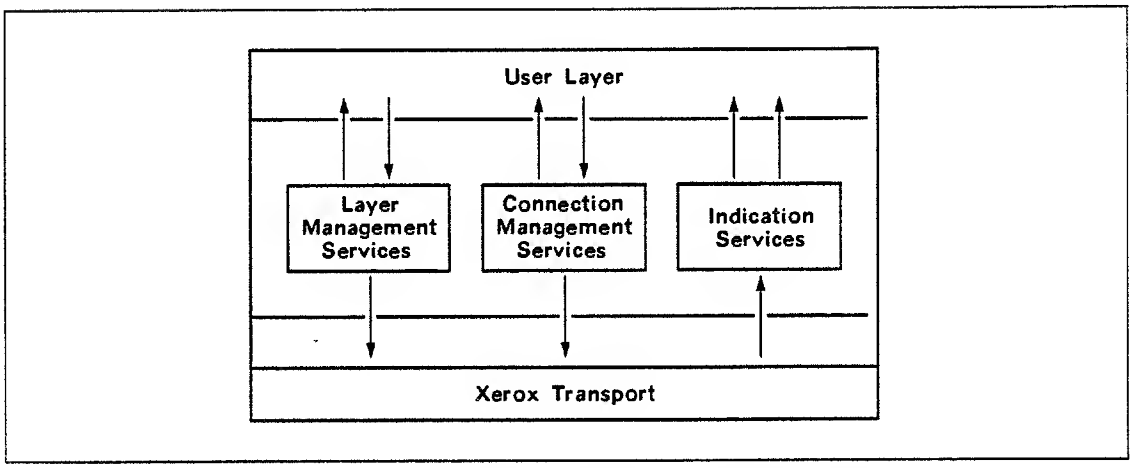


Figure 14-1. Generic Transport Layer Interfaces

Connection Establishment

The connection establishment process starts when a user of the Generic Transport layer initiates a connect request. The Generic Transport layer presents a connect indication to the destination. The destination Generic Transport layer user decides whether or not it can accept this connection. It informs the Generic Transport layer either through a connect response if it can accept a connection, or through a disconnect request if it cannot accept the connection. The Generic Transport layer then presents the results of a connect request to the initiator, either through a connect confirm indication if the connection is made, or through a disconnect indication if the connection is not made.

All transport layer activity can be described in terms of a single point-to-point connection. Each transport connection physically consists of a single transport connection management table, which contains all the information relevant to the connection and which uniquely identifies the connection from all other connections in the Generic Transport layer. The address of this table is referred to as the connection endpoint identifier (CEPID). The CEPID on the user's side is referred to as the USER_CEPID and is used by the Generic Transport layer when sending indications and confirmations. The CEPID on the Generic Transport layer's side is referred to as SERVICE_CEPID. The user provides SERVICE_CEPID in request and response procedures.

Fragmentation and Reassembly of Data

A major responsibility of the Generic Transport layer is to provide fragmentation and reassembly of normal data, because the Xerox Transport layer has an upper limit on the maximum data unit size. The data received from the user is fragmented into a chain of packets, each consisting of a certain number of bytes. (The number of bytes is set by the Xerox Transport layer after it opens an Internet SAP. See chapter 15 in this manual for details.) Similarly, data received from the peer is held by the Generic Transport layer until all packets are received; the message is reassembled in the right order and the packet is then sent to the user.

Flow Control

The Generic Transport layer is responsible for supporting flow control which is implemented by the Xerox Transport layer. Flow control prevents the user from getting flooded with data. The flow of data from a source is controlled on a packet basis. The source may transmit packets up to and including a number specified by the destination. Flow control is further described in chapter 15 of this manual.

Data Transmission Services

The Generic Transport layer provides two classes of data transmission services:

- Normal
- Expedited

Normal Data Transmission

Data packets using the normal transmission services may contain a range of zero bytes to the maximum data length specified by the Generic Transport layer. These data packets are delivered in first in first out (FIFO) sequence, and are subject to flow control mechanisms that are enforced by the destination.

Expedited Data Transmission

Data packets that use the expedited transmission services may contain a maximum of 16 bytes of data per packet. Expedited data takes precedence over any normal data transfer and is delivered immediately, regardless of the sequence number assigned to the data packets. Expedited data transmission can be used to break user protocol deadlocks and for getting around the flow control mechanism. However, if more than eight packets reside in the data transmit and/or acknowledge queues of the Xerox Transport layer, a stop indication is sent by the Xerox Transport layer to the Generic Transport layer. Although the Xerox Transport layer may deliver duplicates of expedited traffic, the Generic Transport layer does not deliver any duplicate data packets.

Priority

In each system, connections can be assigned a high or low priority. Transport layer uses the priority information to decide how it should provide indications to its users. On high priority connections, the Generic Transport layer immediately processes and delivers the indications while on low priority connections, the indications are sent at a later time.

The formats and definitions of the service requests from users and the indications sent by the Generic Transport layer are described in Services Provided, later in this chapter.

Services Required

For the Generic Transport layer to provide services to its own users it, in turn, depends on the services of other entities. This section briefly describes the services of each of these entities.

Directory ME

When the Generic Transport layer is loaded in a system, it registers its title with the Directory ME. A user who wishes to communicate with the Generic Transport layer asks for a directory translation. The directory entry has the following values:

Title:	GENERIC_TRANSPORT
Address:	Local address of type, GT_LAYER_MGMT_CALL
Domain:	Local System

Further details are provided in chapter 5 in this manual.

Xerox Transport Layer

The Xerox Transport layer is provided so that CDCNET can support higher layer protocols like the Virtual Terminal Protocol (VTP). The Xerox Transport layer service is supported by the features of the Xerox Sequenced Packet Protocol. The Xerox Transport layer is responsible for informing the Generic Transport layer of the credit allocation of the corresponding entity. Credit allocation is an indication of the receiver's ability to receive and handle a certain amount of data.

The Generic Transport layer uses the end-of-information (EOM) bit in the data sent to it from the Xerox Transport layer to assemble one or more protocol data units (PDUs) of an appropriate size into user's service data units (SDUs).

The Generic Transport layer depends on the Xerox Transport layer to assign sequence numbers to the packets that it fragments. The sequence numbers are used by the destination system to assure FIFO delivery, to ignore duplicates, and to acknowledge the PDUs received.

The Xerox Transport layer uses timers for the following purposes:

- To initiate retransmission of unacknowledged PDUs.
- To maintain the connection during the time when no user data is being transmitted.
- To detect network failures or unilateral disconnections at the correspondent's end.

See chapter 15 in this manual for details.

Services Provided

This section describes the external services provided by the Generic Transport layer to its users. It describes the major data structures used and the service requests exchanged between the Generic Transport layer and the user. The types of service requests are:

- Layer Management Services
- Connection Management Services
- Indicator Services

Constants and common types and detailed explanations of returned status messages are given in Constants and Common Types, later in this chapter.

Layer Management Services

Layer management services perform the preliminary tasks of opening and closing a Generic Transport layer SAP and establishing a connection.

GT#LAYER_MGMT Procedure

Users interface with the Generic Transport layer through direct calls to the GT#LAYER_MGMT procedure. The calls are made with a single parameter which points to a variant record. The variant record contains a workcode and parameters that are dependent on this workcode. The address of this procedure is obtained by the user through a directory title translation. The following is a description of the layer management request:

```
PROCEDURE
      gt#layer_mgmt_request  (
        VAR request: gt_connection_mgmt_request);
```

The common deck TRDGT contains the layer management request record, the connection management procedure, the indication interfaces, and all the type definitions.

Layer Management Request

The layer management request record is the main data structure used by the Generic Transport layer's users to open or close a Transport SAP and to establish a connection.

This data structure is a variant record defined with fields to support both input and output parameters. The workcode, the Generic Transport layer service SAP identifier and the status parameters are common and are used in all layer management requests. A value specified in the workcode field determines other fields or service requests that are to be used in the layer management request.

Following is a CYBIL description of the layer management request record.

```
TYPE
     gt_layer_mgmt_request = record
       workcode: gt_layer_mgmt_codes,
       service_sapid : gt_sap,
       status : gt_status,
       case gt_layer_mgmt_codes of

       = gt_open_sap =
         open_sap@: record
           user_sapid: ^cell,
           dedicated_sapid: sap_id_type,
           user_layer_mgmt_if: generic_connect_if,
           user_connect_mgmt_if: generic_data_if,
           generic_connect_mgmt_if: gt_connection_mgmt_call,
         recend,

       = gt_connect_request
         connect_request@: record
           user_cepid: ^cell,
           destination: gt_sap,
           credit_window: gt_credit_window_range,
           connect_data: buf_ptr,
           priority: generic_priority,
           service_cepid: ^cell,
           recend,

       = gt_close_sap = no parameters

       casend,
     recend;
```

Following are the common fields defined in the static portion of the layer management request record:

workcode: gt_layer_mgmt_codes

This field must contain one of the following values:

GT_OPEN_SAP

GT_CLOSE_SAP

GT_CONNECT_REQUEST

The user must supply one of these values to specify a service request when making a layer management request. Details on these service requests are given later in this chapter.

service_sapid: gt_sap

This is the SAP identifier of the Generic Transport layer.

status: gt_status

This is the status indication of a processed request returned by the Generic Transport layer. Return codes are listed with each service request later in this chapter. A complete list of status messages with explanations is also given in Constants and Common Types, later in this chapter.

The following pages describe the various service requests offered to a Generic Transport layer user. The user requests a service by specifying an appropriate value in the workcode field of the layer management request record. The common fields described earlier are listed again in the following descriptions of the service requests. The explanation to each common field is not repeated unless there is some unique information for that field in a particular service request.

GT_OPEN_SAP

This request allows the user to open a Generic Transport layer SAP. Through this request, the Xerox Transport layer is notified to open an Internet SAP. During this request, the user passes pointers to two procedures which the Generic Transport layer uses to send confirmations and indications to requests.

A workcode of GT_OPEN_SAP in the layer management request record initiates this request.

Input The user sets up the following fields as input in the layer management request record:

workcode: gt_layer_mgmt_codes

This field must contain the following value:

GT_OPEN_SAP

user_sapid: ^cell

The address of the user's SAP identifier table, which is specified by the Generic Transport layer when it returns an indication to this request.

dedicated_sapid: sap_id_type

The 3B SAP identifier. The value of this parameter is 0 if a dynamically assigned SAP is opened. See chapter 16 in this manual for details on dedicated and nondedicated SAPs.

user_layer mgmt_if: generic_connect_if

The address of the user's layer management interface, which is used by the Generic Transport layer to send connect indications.

user_connect_mgmt_if: generic_data_if

The address of the user's connection management interface, which is used by the Generic Transport layer to send all other indications.

Output The Generic Transport layer returns the following fields as output in the layer management request record:

service_sapid: gt_sap

The address of the Generic Transport layer's SAP that was just opened.

generic_connect_mgmt_if: gt_connection_mgmt_call

The address of the Generic Transport layer's connection management interface to which the user will send all connection management requests.

status: gt_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
gt_no_memory_for_sap
gt_sap_busy
gt_sap_open
```

GT_CLOSE_SAP

This request allows the user to close a currently open SAP. All established tables and connections associated with this SAP are terminated without notifying the user, and incoming connect requests for this SAP are no longer accepted.

A workcode of GT_CLOSE_SAP in the layer management request record initiates this request.

Input The user sets up the following fields as input in the layer management request record:

workcode: gt_layer_mgmt_codes

This field must contain the following value:

GT_CLOSE_SAP

service_sapid: gt_sap

This is the address of the Generic Transport layer's SAP that is to be closed.

Output The Generic Transport layer returns the following field as output in the layer management request record:

status: gt_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
gt_source_sap_not_found
gt_request_processed
```

GT_CONNECT_REQUEST

This request allows the Generic Transport layer user to establish a connection with a peer Generic Transport layer user.

A workcode of GT_CONNECT_REQUEST in the layer management request record initiates this request.

Input The user sets up the following fields as input in the layer management request record:

workcode: gt_layer_mgmt_codes

This field must contain the following value:

GT_CONNECT_REQUEST

service_sapid: gt_sap

user_cepid: ^cell

This is the address of the user's connection table which the Generic Transport layer uses when returning indications and confirmations relevant to this connection.

destination: gt_sap

This is the SAP identifier of the destination Generic Transport layer to which the connection is being requested.

credit_window: gt_credit_window_range

This field supports the flow control mechanism which is enforced by the Xerox Transport layer. Maximum window size is 8 Xerox Transport layer packets. For details on flow control, see chapter 15 in this manual.

user_data: buf_ptr

A pointer to a record called DATA_DESCRIPTOR. This record contains data that is passed on to the destination. Maximum connect data is 32 bytes.

priority: generic_priority

This field indicates the priority used to determine when an indication should be provided to the user. This field contains one of the following values:

LOW The indications are queued and sent later.

HIGH The indications are sent immediately.

Output The Generic Transport layer returns the following fields as output in the layer management request record:

service_cepid: ^cell

This is the address of the Generic Transport layer's connection table.

status: gt_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
gt_message_exceeds_max_length
gt_source_sap_not_found
gt_request_processed
gt_credit_not_within_limits
gt_no_memory_for_connection
```

Connection Management Services

Connection management services manage the activity on connections. Connection management services do the following:

- Accept connections.
- Disengage connections.
- Transfer data.
- Monitor flow control.

GT#CONNECTION_MGMT Procedure

Generic Transport's connection management requests are made by calling the GT#CONNECTION_MGMT procedure. The address of this procedure is passed to the user as an output parameter in an OPEN_SAP request. The connection management request passed in the procedural call is defined below:

```
TYPE

  gt#connection_mgmt_call = ^procedure (
     VAR request: gt_connection_mgmt_request);
```

Connection Management Request

The connection management request is the main data structure used by Generic Transport layer users to manage their data exchange. This data structure is a variant record with fields to support both input and output parameters. The workcode, the address of the Generic Transport layer connection table (CEPID), and the status fields are common and are used in all connection management requests. A value specified in the workcode field determines the other fields or service requests that are to be used in the connection management request.

Following is a CYBIL description of the connection management request record.

```
TYPE
gt_connection_mgmt_request = record
  workcode: gt_connection_mgmt_codes,
  service_cepid: ^cell,
  status: gt_status,
  case gt_connection_mgmt_codes of

     = gt_connect_accept =
       connect_accept@: record
         priority: generic_priority,
         credit_window: gt_credit_window_range,
         accept_data: buf_ptr,
       recend,

    = gt_data_request, gt_xdata_request, gt_disconnect_request=
      user_data: buf_ptr,

  = gt_flow_control_request =
    flow_control_code : gt_flow_control_request_code,

  = gt_abort_request =

    casend,
recend;
```

Following are the common fields defined in the static portion of the connection management request record:

workcode: gt_connection_mgmt_codes

This field must contain one of the following values:

GT_CONNECT_ACCEPT

GT_DATA_REQUEST

GT_XDATA_REQUEST

GT_DISCONNECT_REQUEST

GT_FLOW_CONTROL_REQUEST

GT_ABORT_REQUEST

The user must supply one of these values to specify a service request when making a connection management request. Details on these service requests are given later in this section.

service_cepid: ^cell

The address of the Generic Transport layer's connection table. This is the CEPID of the Generic Transport layer to which the connection is being made.

status: gt_status

The status indication of a processed request returned by the Generic Transport layer. Return codes are listed with each service request. A complete list of status messages with explanations is also given in Constants and Common Types, later in this chapter.

The following pages describe the various service requests offered to a Generic Transport layer user. The user requests a service by specifying an appropriate value in the workcode field of the connection management request record. The common fields described earlier are listed again in the following descriptions of the service requests. The explanation to each common field is not repeated unless there is some unique information for that field in a particular service request.

GT_CONNECT_ACCEPT

This request allows the user to notify the Generic Transport layer that it accepts the connection request that was sent by its peer entity.

A workcode of GT_CONNECT_ACCEPT in the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: gt_connection_mgmt_codes

This field must contain the following value:

GT_CONNECT_ACCEPT

service_cepid: ^cell

accept_data: buf_ptr

A pointer to a record called DATA_DESCRIPTOR. This record contains data that is passed on to the destination. Maximum data passed in this field is 32 bytes.

credit_window: gt_credit_window_range

This field supports flow control which is enforced by the Xerox Transport layer. Maximum window size is 8 Xerox Transport layer packets. For details on flow control, see chapter 15 in this manual.

priority: generic_priority

This field indicates the priority given to this connection.

Output The Generic Transport layer returns the following field as output in the connection management request record:

status: gt_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
gt_credit_not_within_limits
gt_message_exceeds_max_length
gt_invalid_state
gt_connection_not_found
gt_request_processed
```

GT_DATA_REQUEST

This request allows the Generic Transport layer user to send data to its peer on an established connection.

A workcode of GT_DATA_REQUEST in the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: gt_connection_mgmt_codes

This field must contain the following value:

GT_DATA_REQUEST

service_cepid: ^cell

user_data: buf_ptr

A pointer to a buffer chain which contains data that is passed on to the destination. A maximum of 1454 bytes of user data can be passed through this field.

Output The Generic Transport layer returns the following field as output in the connection management request record:

status: gt_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
gt_request_processed
gt_invalid_state
gt_connection_not_found
```

GT_XDATA_REQUEST

This request allows the Generic Transport layer user to exchange expedited data with its peer on an established connection. Expedited data consists of 1 through 16 bytes per PDU, and is not subject to flow control. The data is delivered immediately regardless of the sequence number assigned to the packet.

A workcode of GT_XDATA_REQUEST in the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: gt_connection_mgmt_codes

This field must contain the following value:

GT_XDATA_REQUEST

service_cepid: ^cell

user_data: buf_ptr

A pointer to a buffer chain which contains data that is passed on to the destination. A maximum of 16 bytes of user data can be sent through this field.

Output The Generic Transport layer returns the following field as output in the connection management request record:

status: gt_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
gt_request_processed
gt_invalid_state
gt_connection_not_found
gt_message_exceeds_max_length
```

GT_DISCONNECT_REQUEST

This request allows the Generic Transport layer user to break a connection with a peer Generic Transport layer user. A workcode of GT_DISCONNECT_REQUEST in the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: gt_connection_mgmt_codes

This field must contain the following value:

GT_DISCONNECT_REQUEST

service_cepid: ^cell

user_data: buf_ptr

A pointer to a buffer chain which contains data that is passed on to the destination. A maximum of 64 bytes of user data can be sent through this field.

Output The Generic Transport layer returns the following field as output in the connection management request record:

status: gt_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
gt_request_processed
gt_connection_not_found
gt_message_exceeds_max_length
```

Remarks The Generic Transport layer's disengagement mechanism is an end-to-end disconnect; the local end of the connection is dissolved unilaterally, and the correspondent is notified.

GT_FLOW_CONTROL_REQUEST

This request allows the user to indicate to the Generic Transport layer to either start or stop sending data. Flow control consists of having the peer grant a credit allocation, indicating the amount of data that it guarantees it will be able to receive.

A workcode of GT_FLOW_CONTROL_REQUEST in the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: gt_connection_mgmt_codes

This field must contain the following value:

GT_FLOW_CONTROL_REQUEST

service_cepid: ^cell

flow_control_code: gt_flow_control_request_code

This field has two values:

GT_START_ REQUEST	Indicates to the Transport layer there is no congestion and it can start sending data.
GT_STOP_ REQUEST	Indicates congestion and requests the Transport layer to stop sending data.

Output The Generic Transport layer returns the following field as output in the connection management request record:

status: gt_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
gt_request_processed
gt_connection_not_found
```

GT_ABORT_REQUEST

This request allows the Generic Transport layer user to abort a connection. An abort service, unlike a disconnect service, brings a connection to an abrupt close. The local end of the connection is dissolved. The peer or destination system is not informed of the abort; instead, the Xerox Transport layer in the destination system discovers it through the timing mechanism, and informs the Generic Transport layer through a disconnect indication.

A workcode of GT_ABORT_REQUEST in the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: gt_connection_mgmt_codes

This field must contain the following value:

GT_ABORT_REQUEST

service_cepid: ^cell

Output The Generic Transport layer returns the following field as output in the connection management request record:

status: gt_status

This is the status indication for the processed request. Following are status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
gt_request_processed
gt_connection_not_found
```

Indication Services

Indication services are used by the Generic Transport layer to inform the user that a certain event has occurred. One procedure has been provided to handle only connect related indications and another routine handles all other indications.

Connect Indication

A connect indication informs a Generic Transport layer user that a peer entity across the network has issued a connect request. The user sends either a connect response if the connection is acceptable, or a disconnect indication if the connection is unacceptable. A special case exists where a user immediately detects that a connection is not possible. In this case, a disconnect indication is sent by returning a NIL in the CEPID value.

When a Generic Transport layer receives a connect response from a user, it sends a connect confirm indication to the user that originally issued the connect request.

GENERIC_CONNECT_IF

The Generic Transport layer presents the connect indications to its users by calling the GENERIC_CONNECT_IF procedure. The address of this procedure is passed as an output parameter in an OPEN_SAP request. The indication passed in the procedural call to the user is defined below:

```
TYPE
     generic_connect_if = ^procedure (cepid: generic_cepid;
       VAR sdu: buf_ptr;
       source: gt_sap;
       user_sap: usapid;
       VAR cepid: ucepid);
```

cepid: generic_cepid

This is the CEPID of the Generic Transport layer that is initiating the connection.

sdu: buf_ptr

This parameter contains the address of the chain of buffers which contain the data received in the GT_CONNECT_REQUEST from the peer user. A maximum of 32 bytes of data can be contained in these buffers.

source: gt_sap

This parameter indicates the SAP identifier of the Generic Transport layer.

user_sap: usapid

The SAP identifier of the user to which the connect indication is being sent.

cepid: ucepid

The CEPID of the user receiving the connect indication.

Other Indications

The following procedure is used for all indications and confirmations other than the connect indication.

GENERIC_DATA_IF

The Generic Transport layer presents all indications except the connect indications to its users by calling the GENERIC_DATA_IF procedure. The address of this procedure is passed as an output parameter in an OPEN_SAP request. The indication passed in the procedural call to the user is defined below:

```
TYPE
     generic_data_if = ^procedure (interface: indgeneric;
       cepid: ucepid;
       VAR sdu: buffer);
```

interface: indgeneric

This parameter indicates the type of indication that the Generic Transport layer is sending to the user. The following is a list of values for this parameter:

CONNECT_CONFIRM

DISCONNECT_INDICATION

DATA_INDICATION

XDATA_INDICATION

START_INDICATION

STOP_INDICATION

START_XDATA_INDICATION

STOP_XDATA_INDICATION

cepid: ucepid

This parameter is the Generic Transport layer's CEPID for the connection for which an indication is provided.

sdu: buf_ptr

This parameter refers to the chain of buffers.

The following pages describe the various indications that the Generic Tranport layer sends to its users.

CONNECT_CONFIRM

This indication allows the Generic Transport layer to inform its users that the connection request it had sent out earlier was accepted by its peer.

A value of CONNECT_CONFIRM in the GENERIC_DATA_IF procedure initiates this request.

Input The Generic Transport layer sets up the following parameters as input in the GENERIC_DATA_IF procedure:

interface: indgeneric

This parameter must contain the following value:

CONNECT_CONFIRM

cepid: ucepid

sdu: buf_ptr

DISCONNECT_INDICATION

This indication is used by the Generic Transport layer to inform its users about the termination of a connection. This indication could be sent to inform a user that the connection request it had sent out earlier was not accepted by its peer, or a peer user terminated a connection after a data transfer, or a number of other reasons.

A value of DISCONNECT_INDICATION in the GENERIC_DATA_IF procedure initiates this request.

Input The Generic Transport layer sets up the following parameters as input in the GENERIC_DATA_IF procedure:

interface: indgeneric

This parameter must contain the following value:

DISCONNECT_INDICATION

cepid: ucepid

sdu: buf_ptr

DATA_INDICATION

This indication allows the Generic Transport layer to inform its users that there is data for a user on a connection.

A value of DATA_INDICATION in the GENERIC_DATA_IF procedure initiates this request.

Input The Generic Transport layer sets up the following parameters as input in the GENERIC_DATA_IF procedure:

interface: indgeneric

This parameter must contain the following value:

DATA_INDICATION

cepid: ucepid

sdu: buf_ptr

XDATA_INDICATION

This indication allows the Generic Transport layer to inform its users that there is expedited data for a user on a connection.

A value of XDATA_INDICATION in the GENERIC_DATA_IF procedure initiates this request.

Input The Generic Transport layer sets up the following parameters as input in the GENERIC_DATA_IF procedure:

interface: indgeneric

This parameter must contain the following value:

XDATA_INDICATION

cepid: ucepid

START_INDICATION

This indication allows the Generic Transport layer to inform its users that there is no congestion on the connection and the user can start transmitting data again.

A value of START_INDICATION in the GENERIC_DATA_IF procedure initiates this request.

Input The Generic Transport layer sets up the following parameters as input in the GENERIC_DATA_IF procedure:

interface: indgeneric

This parameter must contain the following value:

START_INDICATION

cepid: ucepid

STOP_INDICATION

This indication allows the Generic Transport layer to inform its users to stop sending the data because the connection is congested and there is no credit allocated to transmit the data.

A value of STOP_INDICATION in the GENERIC_DATA_IF procedure initiates this request.

Input The Generic Transport layer sets up the following parameters as input in the GENERIC_DATA_IF procedure:

interface: indgeneric

This parameter must contain the following value:

STOP_INDICATION

cepid: ucepid

START_XDATA_INDICATION

This indication allows the Generic Transport layer to inform its users that there is no congestion on the connection and the user can start transmitting expedited data again.

A value of START_XDATA_INDICATION in the GENERIC_DATA_IF procedure initiates this request.

Input The Generic Transport layer sets up the following parameters as input in the GENERIC_DATA_IF procedure:

interface: indgeneric

This parameter must contain the following value:

START_XDATA_INDICATION

cepid: ucepid

STOP_XDATA_INDICATION

This indication allows the Generic Transport layer to inform its users to stop sending expedited data because the connection is congested and there is no credit allocated to transmit the data.

A value of STOP_XDATA_INDICATION in the GENERIC_DATA_IF procedure initiates this request.

Input The Generic Transport layer sets up the following parameters as input in the GENERIC_DATA_IF procedure:

interface: indgeneric

This parameter must contain the following value:

STOP_XDATA_INDICATION

cepid: ucepid

Constants and Common Types

This section lists the constants and common types used in the Generic Transport layer's service requests. It also lists and explains status messages returned for the processed requests. These data types are defined in the common deck TRDGT.

Constants

```
gt_layer_mgmt_title = 'generic_transport'
gt_max_credit_window = 8
```

Common Types

```
generic_cepid = ^cell

generic_priority = (low, high)

gt_connection_mgmt_codes = (
        gt_connect_accept, gt_data_request, gt_xdata_request,
        gt_disconnect_request,
       gt_flow_control_request, gt_abort_request)

gt_credit_window_range = 1 .. gt_max_credit_window

gt_flow_control_request_code = (gt_start_request, gt_stop_request)

gt_layer_mgmt_codes = ( gt_open_sap, gt_close_sap, gt_connect_request)

gt_sap = internet_address

gt_status =
    (gt_request_processed,
    gt_credit_not_within_limits,
    gt_source_sap_not_found,
    gt_message_exceeds_max_length,
    gt_invalid_state,
    gt_sap_open,
    gt_sap_busy,
    gt_no_memory_for_sap,
    gt_connection_not_found,
    gt_no_memory_for_connection)
```

gt_request_processed	The request was processed without any errors.
gt_credit_not_within_limits	The credit window is invalid.
gt_source_sap_not_found	The specified SAP does not exist.
gt_message_exceeds_max_length	The message exceeds the maximum length for a particular type of request.
gt_invalid_state	Transport is not in the expected state.
gt_sap_open	The request to open SAP was successful.
gt_sap_busy	The specified SAP is already opened.
gt_no_memory_for_sap	There is not enough memory for SAP table.
gt_connection_not_found	The connection does not exist.
gt_no_memory_for_connection	There is not enough memory for connection table.

```
indgeneric = (connect_indication, connect_confirm, disconnect_indication,
              data_indication, xdata_indication, start_indication,
              stop_indication)

ucepid = ^cell

usapid = ^cell
```

Xerox Transport Layer 15

Xerox Transport Layer 15

This chapter discusses:

- An overview of the Xerox Transport layer.
- The services required by the Xerox Transport layer.
- The services provided by the Xerox Transport layer.

Overview

The Xerox Transport layer is part of the Transport layer (layer 4) and is responsible, along with the Generic Transport layer, for providing end-to-end, transparent data transfer between two users.

The Xerox Transport layer offers the following services:

- It guarantees its users that the message from a source will be delivered to the destination without any errors or duplications and in the same order as it was sent.
- It is responsible for end-to-end flow control.

The Xerox Transport layer is mostly used by the Generic Transport layer. The Generic Transport layer and the Xerox Transport layer are defined in the same module and share many procedures and type declarations. In the current release, no external program interfaces to the Xerox Transport layer have been defined. All Transport layer users interface through the Generic Transport layer.

Xerox Transport Layer Services

Xerox Transport layer services are functionally divided into three groups (figure 15-1):

- Layer management services
- Connection management services
- Indication Services

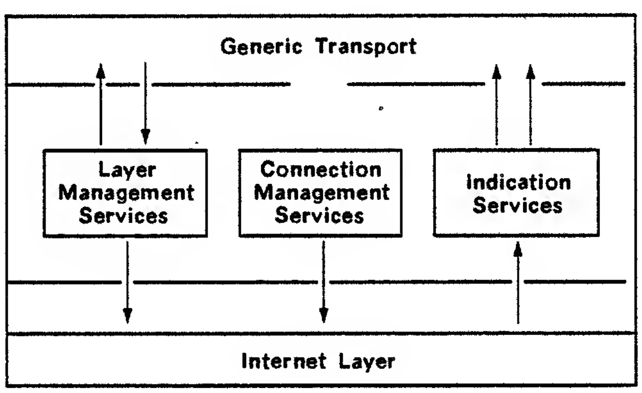


Figure 15-1. Xerox Transport Layer Interfaces

User and Service Interfaces

As shown in figure 15-1, the Xerox Transport layer can be viewed as having three sets of user interfaces and three sets of service interfaces.

User interface groups provide users with the layer management and connection management services such as:

- Opening SAPs
- Establishing connections
- Transferring data on established connections
- Breaking a connection when it is no longer required
- Passing indications and confirmations.

Service interfaces are between the Internet 3B layer and the Xerox Transport layer, and provide the means by which two Transport layers communicate.

Since the Generic Transport layer and the Xerox Transport layer are both defined in the same module, the Generic Transport layer (the Xerox Transport layer's user) directly calls the Xerox Transport layer's open SAP procedure. Once a SAP is opened, a connection is requested. A SAP may accept many incoming connections. Closing a connection does not affect the SAP associated with the connection, but closing the SAP disconnects all the active connections on the SAP that was closed.

Connection Establishment

The connection establishment process starts when the Generic Transport layer initiates a connect request. The Xerox Transport layer presents a connect indication to the destination system. The destination Xerox Transport layer user decides whether or not it should accept this connection. It then informs the Xerox Transport layer through a connect response if it can accept a connection, or through a disconnect request if it cannot accept the connection. The Xerox Transport layer then presents the results of a connect request to the initiator, either through a connect confirm if the connection is made, or through a disconnect indication if the connection is not made.

Transport Connections

All transport layer activity can be described in terms of a single point-to-point connection. Each connection physically consists of a single transport connection management table, which contains all the information relevant to the connection and which uniquely identifies the connection from all other connections in the Xerox Transport layer. The address of this table is referred to as the connection endpoint identifier (CEPID). The CEPID on the user's side is referred to as the USER_CEPID and is used by the Xerox Transport layer when sending indications and confirmations. The CEPID on the Xerox Transport layer side is referred to as XEROX_CEPID. The user provides XEROX_CEPID in request and response procedures.

Each end of the connection takes responsibility for detecting and amending the effects of intermediate network failure and ensuring that a connection is not

broken by the loss of an intermediate system or any other network failure. Each end also takes responsibility to protect itself against the failure of its correspondent, its user, and itself. It makes sure that the loss of previous connections does not affect the current connection.

Priority

In each system, connections can be assigned a high or low priority. The Transport layer uses the priority information to decide how it should provide indications to its users. On high priority connections, the Generic Transport layer immediately processes and delivers the indications while on low priority connections, the indications are sent at a later time.

Sequence Numbers

When a Xerox Transport layer receives data packets from the Generic Transport layer, it assigns sequence numbers to these packets. These numbers are used by the destination Xerox Transport layer to assure FIFO delivery, to detect and discard duplicates, and to acknowledge received packets.

Flow Control

The Xerox Transport layer is also responsible for implementing flow control. Flow control prevents the receiver from getting flooded with data. When a destination Xerox Transport layer user experiences congestion, it requests, through the flow control request procedure, that the Xerox Transport layer stop sending data.

Credit Allocation

Credit allocation is another method of controlling data flow. The destination Xerox Transport layer indicates the number of packets it can handle through credit allocation and credit windows. The Xerox Transport layer periodically updates its credit allocation by indicating, in the header of a packet, the maximum sequence number it can handle from its peer. Packets that exceed this number are discarded. The number by which the Xerox Transport layer increments the allocation depends on credit windows and the memory and buffer state of the system.

Data Transmission Services

The Xerox Transport layer provides two classes of data transmission services:

- Normal data transmission service
- Expedited data transmission service

Normal Data Transmission

Data packets that use the normal transmission services may contain a range from zero bytes to the maximum data length specified by the Internet layer. These data packets are delivered in FIFO sequence, and are subject to flow control mechanisms that are enforced by the destination.

Expedited Data Transmission

Data packets that use the expedited transmission services may contain a maximum of 16 bytes of data per packet. Expedited data takes precedence over any normal data transfer; it is delivered immediately, regardless of the sequence number assigned to the data packets.

Services Required

For the Xerox Transport layer to provide services to its own users it, in turn, depends on the services of the other software components. Brief descriptions of the services of each of these software components follow.

Internet Layer

The following Internet layer interfaces are used by the Xerox Transport layer:

OPEN_INTERNET_SAP	All outbound connection requests are always multiplexed into the Internet SAP of the same number as the Xerox Transport layer SAP. Incoming connections are received at the multiplexed SAP, but are switched to the transient Internet layer SAPs before an indication is sent to the user. The transient SAP is opened by the Xerox Transport layer and remains open for the lifetime of the connection. The Internet layer is responsible for giving the Xerox Transport layer information on the maximum length of data that can be handled on all connections on the SAP that has just been opened. If PACKET_DATA_LIMIT exceeds the value that results from subtracting the length of the transport header from the maximum length of data returned by the Internet layer, then PACKET_DATA_LIMIT is set to this value.
CLOSE_INTERNET_SAP	This interface is used when an Internet layer SAP is no longer required. Both transient and multiplexed SAPs are closed. When a SAP is closed, all the active connections within the SAP are also disconnected.

The Internet layer is also responsible for sending and receiving datagrams. See chapter 16 in this manual for further information on the Internet layer.

Memory Management

Memory Manager keeps the Xerox Transport layer informed of the status of the system memory and the buffers. When the Xerox Tranport layer is informed that memory has to be released, each connection is examined and a connection is chosen to be released based on variables such as the state of the connection and the messages queued. See the CDCNET Systems Programmer's Reference Manual, Volume 1, for more details on memory management.

Services Provided

This section defines the external services provided by the Xerox Transport layer to its users. It describes the major data structure used, and the service requests exchanged between the user and the Xerox Transport layer.

The types of services provided are:

- Layer management services
- Connection management services
- Indications services

Common types used by the Xerox Transport layer and detailed explanations of returned status messages are given in chapter 14 of this manual.

Layer Management Services

Layer management services perform the following preliminary tasks:

- Opening a Xerox Transport layer SAP.
- Closing a Xerox Transport layer SAP.
- Requesting a connection to be opened.

The open SAP request creates a Transport layer SAP that is mapped into an Internet layer SAP.

NOTE

In the current release, no externally defined interfaces to the Xerox Transport layer have been defined. All Transport layer users have to go through the Generic Transport layer.

The following pages describe the Xerox Transport layer management service requests.

XEROX_OPEN_SAP

This request allows the user to open a Xerox Transport layer SAP and an Internet layer 3B SAP. This request is directly passed on to the Internet layer. The Internet layer address corresponding to the SAP that has just been opened is passed back to the user. During this request, the user passes pointers to two procedures which the Xerox Transport layer uses to send connect and data-related indications.

Comdeck **TRMXPRT**

Format **XEROX_OPEN_SAP (usapid, connect, data, dedicated_sap_id, sap, status)**

Input

usapid: usapid

The user's SAP identifier.

connect: connect_if

The address of the user's connect indication interface routine, which is used by the Xerox Transport layer to send connect indications. For details, see GENERIC_CONNECT_IF in chapter 14 of this manual.

data: data_if

The address of the user's data indication interface routine, which is used by the Xerox Transport layer to send data indications. For details, see GENERIC_DATA_IF in chapter 14 of this manual.

dedicated_sap_id: sap_id_type

The identifier of the SAP that is being opened. If the SAP identifier is well known, the dedicated SAP identifier is specified; otherwise, it is specified to zero.

Output

sap: xerox_sap

The Internet layer address. See appendix B in this manual for more information.

status: gt_status

The status indication of the processed request, returned by the Xerox Transport layer. Following are status messages applicable to this request. For explanations, see Constants and Common Types in chapter 14 of this manual.

```
gt_no_memory_for_sap
gt_sap_busy
gt_sap_open
```

XEROX_CLOSE_SAP

This request allows the user to close a Xerox Transport layer SAP. All connections on the SAP are closed and the SAP is released.

Comdeck TRMXPRT

Format XEROX_CLOSE_SAP (sap, status)

Input **sap: xerox_sap**

The address of the SAP that is to be closed.

Output **status: gt_status**

The status indication of the processed request, returned by the Xerox Transport layer. Following are status messages applicable to this request. For explanations, see Contants and Common Types in chapter 14 of this manual.

```
gt_source_sap_not_found
gt_request_processed
```

XEROX_CONNECT

This request allows the user to establish a connection with a peer. If data is included in this request, that data is delivered with the connect indication to the destination system.

Comdeck **TRMXPT**

Format **XEROX_CONNECT (user_cepid, message, data_stream_type, from_sap, to_sap, priority, cepid, xdata_flow_control, credit_window, status)**

Input

user_cepid: user_cepid

The user's CEPID.

message: buf_ptr

Refers to the chain of buffers containing user data. A NIL value indicates no data is included.

data_stream_type: xns$data_stream_type

This value is passed as a user field in the transport header. It is one byte of information carried and delivered with the data and indicates to the peer the type of data in the packet.

from_sap: xerox_sap

The transport SAP from which connection is being established.

to_sap: xerox_sap

The transport SAP to which connection is being made.

priority: xpriority

This parameter indicates the priority at which the incoming data is to be processed.

xdata_flow_control: boolean

This parameter indicates if flow control needs to be exercised. It contains one of the following values:

TRUE	Flow control is exercised.
FALSE	Flow control is not exercised. An expedited data packet may be sent when a preceding packet has not been acknowledged. See XEROX_XDATA later in this chapter for more information.

credit_window: gt_credit_window_range

This parameter indicates the credit window for the connection.

Output

cepid: xerox_cepid

The Xerox Transport layer CEPID.

status: gt_status

The status indication of the processed request, returned by the Xerox Transport layer. Following are status messages applicable to this request. For explanations, see Constants and Common Types in chapter 14 of this manual.

```
gt_source_sap_not_found
gt_request_processed
gt_credit_not_within_limits
gt_no_memory_for_connection
```

Connection Management Services

Connection management services manage the activity on connections.

Connection management services include:

- Accepting connections
- Disengaging connections
- Transferring data
- Monitoring flow control

The Xerox Transport layer uses the Generic Transport layer's procedures. For details, see GT#CONNECTION_MGMT_REQUEST in chapter 14 of this manual.

The following pages describe the Xerox Transport layer connection management service requests.

XEROX_ACCEPT

This request allows the user to notify the Xerox Transport layer that it accepts the connection request that was sent by its peer.

Comdeck **TRMXPRT**

Format **XEROX_ACCEPT (cepid, user_cepid, priority, message, data_stream_type, xdata_flow_control, credit_window, status)**

Input

cepid: xerox_cepid

The Xerox Transport layer CEPID.

user_cepid: ucepid

The user CEPID.

priority: xpriority

This parameter indicates the priority at which the incoming data is to be processed.

message: buf_ptr

Refers to the chain of buffers containing data. A NIL value indicates no data is included.

data_stream_type: xns$data_stream_type

This value is passed as a user field in the Transport layer header. It is one byte of information carried and delivered with the data, and indicates to the peer the type of data in the packet.

xdata_flow_control: boolean

This parameter indicates if flow control needs to be exercised. It contains one of the following values:

TRUE	Flow control must be exercised.
FALSE	An expedited data packet may be sent when a preceding packet has not been acknowledged. See XEROX_XDATA later in this chapter for more information.

credit_window: gt_credit_window_range

This value, along with information on memory and buffer state of the system, is used by the Xerox Transport layer to allocate credits to the peer.

Output

status: gt_status

The status indication of the processed request, returned by the Xerox Transport layer. Following are status messages applicable to this request. For explanations, see Constants and Common Types in chapter 14 of this manual.

```
gt_invalid_state
gt_request_processed
gt_connection_not_found
gt_credit_not_within_limits
```

XEROX_DISCONNECT

This request allows the user to break a connection with a peer. The peer is notified of this request and the connection is closed. This is not a graceful close service; data may be lost. The user can send up to 64 bytes of data with this request.

Comdeck TRMXPRT

Format XNS_DISCONNECT (cepid, message, status)

Input **cepid: xerox_cepid**

The Xerox Transport layer CEPID.

message: buf_ptr

This parameter contains the reason the connection was broken.

Output **status: gt_status**

This is the status indication of the processed request, returned by the Xerox Transport layer. Following are status messages applicable to this request. For explanations, see Constants and Common Types in chapter 14 of this manual.

```
gt_request_processed
gt_connection_not_found
```

XEROX_ABORT

This request allows the user to break a connection with a peer. The peer is not notified that the connection has been terminated. When the peer Transport entity detects that no data has been received from its correspondent, it sends a disconnect indication to its user.

Comdeck TRMXPRT

Format XEROX_ABORT (cepid, return_status)

Input **cepid: xerox_cepid**

The Xerox Transport layer CEPID.

Output **status: gt_status**

The status indication of the processed request returned by the Xerox Transport layer. Following are status messages applicable to this request. For explanations, see Constants and Common Types in chapter 14 of this manual.

```
gt_request_processed
gt_connection_not_found
```

XEROX_DATA

This request allows the Xerox Transport layer user to send data to its peer on an established connection. Up to 1454 bytes can be transmitted through this request.

Comdeck TRMXPRT

Format XEROX_DATA (cepid, message_list, data_stream_type, eom, status)

Input

cepid: xerox_cepid

The Xerox Transport layer CEPID.

message_list: buf_ptr

Pointer to the chain of buffers containing data. Data in each chain can range from 0 through 1454 bytes. A NIL value indicates empty buffers.

data_stream_type: xns$data_stream_type

This value is passed as a user field in the Transport header. It is one byte of information carried and delivered with the data, and indicates to the peer the type of data in the packet.

eom: boolean

This parameter indicates the end of data. The end of message flag is set to one of the following values:

TRUE	This is the last data packet.
FALSE	More data packets follow this data packet.

Output

status: gt_status

This is the status indication of the processed request returned by the Xerox Transport layer. Following are status messages applicable to this request. For explanations, see Constants and Common Types in chapter 14 of this manual.

```
gt_request_processed
gt_connection_not_found
gt_invalid_state
```

XEROX_XDATA

This request allows the Xerox Transport layer user to send priority data to its peer on an established connection. Up to 16 bytes can be transmitted through this request.

Expedited data takes precedence over any normal data transfer. Expedited data is generally not subjected to flow control mechanisms; however, flow control can be imposed by passing a parameter in a connect request (see XEROX_CONNECT earlier in this chapter).

Comdeck **TRMXPRT**

Format **XEROX_XDATA (cepid, message_list, data_stream_type, eom, status)**

Input

cepid: xerox_cepid

The Xerox Transport layer CEPID.

message_list: buf_ptr

Pointer to the chain of buffers containing data. Data can range from 0 through 16 bytes. A NIL value indicates empty buffers.

data_stream_type: xns$data_stream_type

This value is passed as a user field in the transport header. It is one byte of information carried and delivered with the data, and indicates to the peer the type of data in the packet.

Output

status: gt_status

This is the status indication of the processed request returned by the Xerox Transport layer. Following are status messages applicable to this request. For explanations, see Constants and Common Types in chapter 14 of this manual.

```
gt_request_processed
gt_connection_not_found
gt_invalid_state
```

XEROX_FLOW_CONTROL

This request allows the user to indicate to the Xerox Transport layer to either start or stop sending data.

Flow control consists of having the peer entity grant a credit allocation, indicating the amount of data that it guarantees it will be able to receive.

Comdeck TRMXPRT

Format **XEROX_FLOW_CONTROL (cepid, flow_control_code, status)**

Input

cepid: xerox_cepid

The Xerox Transport layer CEPID.

flow_control_code: gt_flow_control_request_code

This parameter indicates whether the user should start or stop sending data. It contains one the following values:

GT_START_REQUEST	Indicates to the Xerox Transport layer that there is no congestion and that it can start sending data.
GT_STOP_REQUEST	Indicates congestion and requests that the Xerox Transport layer stop sending data.

Output

status: gt_status

The status indication of the processed request returned by the Xerox Transport layer. Following are status messages applicable to this request. For explanations, see Constants and Common Types in chapter 14 of this manual.

```
gt_request_processed
gt_connection_not_found
```

Indications Services

Indications services provide a means for the Xerox Transport layer to inform the user that a certain event has occurred. The Xerox Transport layer uses the same indication routines that are used by the Generic Transport layer. See chapter 14 in this manual for details.

Constants and Common Types

Since the Generic Transport layer and the Xerox Transport layer are defined in the same module, the Xerox Transport layer uses the same common types as the Generic Transport layer. See chapter 14 for details.

Network Layer

Control Data Network Architecture (CDNA) has divided the OSI layer 3 (the Network layer) into the following two sublayers:

- Internet layer (layer 3B)
- Intranet layer (layer 3A)

Internet Layer

The Internet layer (layer 3B) is responsible for sending datagrams from a source to a destination. The datagrams may pass through several systems across various network solutions.

Intranet Layer

The Intranet layer (layer 3A) is responsible for routing and relaying information within a specific network.

The following two chapters describe the Internet (3B) and Intranet layers (3A).

Internet Layer 16

Internet Layer 16

This chapter discusses:

- An overview of the Internet layer.
- The services required by the Internet layer.
- The services provided by the Internet layer.
- Constants and common types used in the Internet layer.

Overview

The Internet layer is a middle layer software component. The Internet layer is used by the Transport layer and network management entities. CDCNET's Internet uses the XEROX Internet Transport Protocol, to provide its services.

The Internet layer provides the following services:

- Provides datagram services to a corresponding Internet layer

  The Internet layer is responsible for delivering datagrams from a source to a destination. It takes messages called packets from its users and delivers them to a destination system.

  If the destination is a remote system, Internet layers in intermediate systems keep relaying the packet forward to the next system until it reaches the system it is intended for.

  Datagram services make every attempt to deliver a packet, but do not provide any guarantees against these packets being lost, duplicated, or delivered out of sequence. There is, therefore, a certain risk in allowing the MEs and applications to interface directly to the Internet layer instead of the Transport layer, which can guarantee end-to-end delivery.

- Provides relaying and routing services

  Delivering data and determining paths on which the data can be delivered are important services of the Internet layer. In other words, the Internet layer is responsible for routing data through the network. To perform the routing functions, the Internet layer depends on the services of:

  - The Intranet layer to transmit and receive data units from directly-connected network solutions.
  - The Routing ME to maintain the routing data stores.

  The Internet layer can receive data units from two sources:

  - From within the CDCNET system in which it resides.
  - From outside the system across a directly-connected network solution.

When an Internet layer user sends a request to the Internet layer to deliver a data unit to another system, it includes the network address of the destination system. The Internet layer routes the data units based on this destination network address, which consists of a network identifier system identifier and 3B SAP identifier. (See chapter 2 and appendix B in this manual for more information on network address and SAPs.)

A data unit that originated from a remote system goes through the following three routing steps before it reaches the destination for which it is intended.

Internetwork Routing

This step ensures that the data unit reaches the network solution whose network identifier is the same as that of the destination system. This is done by delivering the data unit to any system that is directly connected to the desired network solution. The path to another network is a sequence of links or connected networks, in which the first link in the sequence connects to the source and the last link connects to the destination.

The Internet layer selects the paths based on:

- The configuration of networks.

- The cost that is assigned to each path.

The Internet layer determines the next hop in the path to the destination, based on the cost of each path. The cost of each path is supplied to the Internet layer by the Routing ME, and is determined by an algorithm that is based on line speed and line congestion factors. See chapter 4 in this manual for more information on Routing ME.

Each hop is a separate routing decision, and each time the Internet layer relays data, it increments the hop count in the Internet header of the data unit. If the resultant hop count exceeds 16, the Internet layer discards the data unit and informs the sender or the originator of the message.

Intranetwork Routing

In this step, the data unit is delivered to the appropriate system on the destination network solution. The system identifier in the destination address is used to locate the desired system.

Intrasystem Routing

The data unit is finally delivered to the appropriate software component. The 3B SAP identifier is used to accomplish this routing.

- Detects errors

  The Internet layer detects errors and informs Error ME about them. See chapter 11 in this manual for the types of errors and other details on Error ME.

- Relieves users from needing to know the nature of subnetworks

  The Internet layer provides its users with enhanced services that are based on the services provided by the lower layers. Because of this support, its users can continue with their activities without any knowledge of the characteristics or the protocols of the lower layers.

- Supports multicast service

  The Internet layer supports multicast service, a service through which a set of systems with a multicast address can automatically have data delivered to them. See chapter 1 in this manual for a definition of a multicast address.

Services Required

For the Internet layer to provide services to its own users it, in turn, depends on the services of the other software components. Brief descriptions of the services of each of these software components follow.

Executive Interface

The Internet layer uses the interface to the Buffer Management service within the Executive. Details on this interface are provided in the CDCNET Systems Programmer's Reference Manual, Volume 1.

Error ME

When an error is detected, the Internet layer sends the 3B PDU and error code to Error ME through an intertask message (ITM). See chapter 11 in this manual for more information.

Intranet Layer (Layer 3A)

The Internet layer communicates with the Intranet layer (3A) through two interfaces:

USER_DATA_IND_PROC	Used by the Intranet layer to indicate that data has arrived for the Internet layer.
DATA_REQUEST_3A	Used by the Internet layer to send data to the Intranet layer.

See chapter 17 in this manual for details on the Intranet layer.

Services Provided

This section describes the external interfaces provided by the Internet layer to its users.

The Internet layer users include the common deck B3XREQI in their calling modules. This common deck defines the externally referenced (XREF) declarations for the Internet layer procedures.

Common deck B3XREQI contains calls to several common decks which define the types and constants for these procedures and the different data structures through which Internet layer requests are passed.

Common types used by the Internet layer and detailed explanations of returned status messages are given under Constants and Common Types, later in this chapter.

The Internet layer's services are provided through the following procedures:

- OPEN_INTERNET_SAP
- CLOSE_INTERNET_SAP
- Data Request Procedure
- Data Indication Procedure

Figure 16-1 illustrates these interfaces.

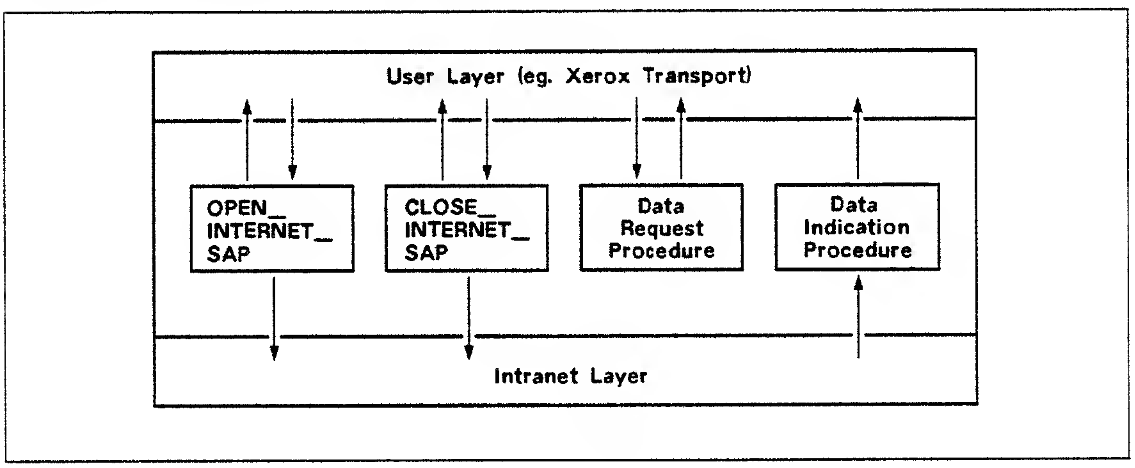


Figure 16-1. Internet Layer Interfaces

OPEN_INTERNET_SAP

This request opens an Internet SAP. A dedicated or dynamic SAP can be opened. The user identifies the SAP it wishes to open and supplies all the input parameters, including the address of the procedure which receives indications from the Internet layer. The Internet layer returns the open SAP identifier along with the address of the procedure to which the user sends data requests.

Comdeck **B3XREQI**

Format **OPEN_INTERNET_SAP (input_param, output_param, return_code)**

Input **input_param: ^open_sap_input_parameters**

A record. Table 16-1 shows this record's input parameters.

Table 16-1. Open SAP Input Record (OPEN_SAP_INPUT_PARAMETERS)

Field	Content
sap_id	The identifier of the SAP the user wishes to open. A zero value indicates a dynamic SAP. (Type SAP_ID_TYPE.)
user_id	A pointer to a cell which contains the user identifier. The Internet layer uses this when delivering data. This pointer is passed again when the user wishes to close the SAP. The user identifier must match the SAP identifier that was passed when the SAP was opened. This prevents accidental closing of SAPs. (Type ^CELL.)
destination	A pointer to the procedure that receives indications from the Internet layer. (Type DESTINATION_3B_SAP_IF.) See Data Indication Procedure and table 16-2 in this chapter for more information on this procedure.
force_close	A pointer to a procedure that is called by the Internet layer to close a SAP. (Type FORCE_CLOSE_IF.) See Constants and Common Types later in this chapter for more information on this procedure.

Output **output_param: ^open_sap_output_parameters**

A record. Table 16-2 shows this record's output parameters.

Table 16-2. Open SAP Output Record (OPEN_SAP_OUTPUT_PARAMETERS)

Field	Content
local_internet_address:	This field is returned by the Internet layer and indicates the network and system address and the SAP identifier of the local system. (Type INTERNET_ADDRESS.) See appendix B for a description of the Internet address record.
internet_request:	A pointer to a procedure which is called by the user to request datagram transmission. (Type INTERNET_REQUEST_ADDRESS.) See Data Request Procedure and table 16-3 for more information.
maximum_request_length	This field is returned by the Internet layer and indicates the maximum number of bytes that can be transmitted (1 through the value of MAX_DATA_LENGTH, 1466 bytes).

return_code: open_internet_sap_status

The status indication for the processed request. Following are the status messages applicable to the OPEN_INTERNET_SAP request. For explanations, see Constants and Common Types later in this chapter.

```
open_sap_successful
illegal_dedicated_sap
nil_parameter_pointer
no_destination_proc
sap_already_opened
no_sap_entries_available
```

Data Request Procedure

This procedure allows the user to request the Internet layer to transmit a data unit. The address of this procedure is returned by the Internet layer when an OPEN_INTERNET_SAP call is made. The parameters for the request are contained in a record whose pointer is passed in the call. (See table 16-3.)

```
TYPE
 internet_request_address = ^procedure (
      req_param: ^internet_req_if;
 VAR return_code: internet_return_codes),
```

Table 16-3. Data Request Record (INTERNET_REQ_IF)

Field	Content
source_address	This field indicates the address of the source or the sender of the message. It includes the network, system, and SAP identifiers. Default values can be supplied for the network and system identifiers. These are listed in Constants and Common Types later in this chapter. Default values specify a local system. See appendix B for a description of the INTERNET_ADDRESS. (Type INTERNET_ADDRESS.)
destination_address	This field indicates the address of the destination system to which the data has to be delivered. It includes the network, system, and SAP identifiers. Default values can be supplied for the network and system identifiers. These values are listed in Constants and Common Types later in this chapter. Default values specify a local system. See appendix B for a description of INTERNET_ADDRESS. (Type INTERNET_ADDRESS.)
packet_kind	This field indicates the protocol identifier to be used in the header of the 3B PDU. It is supplied by the user that is sending the data, and is delivered to the peer at the destination system. A list of known values for packet types for Internet requests and indications is given in Constants and Common Types later in this chapter. (Type PACKET_TYPE.)
checksum	This field indicates if checksum is specified. (Type BOOLEAN.) TRUE — The checksum of the message is calculated, included in the header, and verified at the destination. FALSE — Checksum not specified.
data	A pointer to the data that is being transmitted. The maximum data length is 1466 bytes. (Type BUF_PTR.)

return_code: internet_return_codes

The status indication of a processed request returned by the the Internet layer. Following are the status messages returned for the data request procedure (INTERNET_REQ_IF). For explanations, see Constants and Common Types later in this chapter.

```
internet_success
ineterror_nil_param
ineterror_sosap
ineterror_dssap
ineterror_data
```

Data Indication Procedure

This procedure allows the user to receive data indications. The address of this procedure is specified by the user when an OPEN_INTERNET_SAP call is made. The parameters for the procedure are contained in a record whose pointer is passed in the call. All fields in this record, except the multicast field, are extracted from the Internet PDU headers and passed on to the user. See table 16-4.

```
TYPE
 destination_3b_sap_if = ^procedure (
     ind_params: ^internet_ind_if),
```

Table 16-4. Data Indication Record (INTERNET_IND_IF)

Field	Content	
multicast	The field that indicates if the data is to be multicast. (Type BOOLEAN.)	
	TRUE	Data was received through a multicast service.
	FALSE	Data was received through a datagram service.
checksum	This indicates if checksum is specified. (Type BOOLEAN.)	
	TRUE	The checksum of the message is calculated, included in the header, and verified at the destination.
	FALSE	Checksum is not specified.
source_address	This parameter indicates the address of the source or the sender of the message. It includes the network, system, and SAP identifiers. These are listed in Constants and Common Types later in this chapter. (Type INTERNET_ADDRESS.) See appendix B for a description of the INTERNET_ADDRESS.	
destination_address	This parameter indicates the address of the destination system to which the data has to be delivered. It includes the network, system, and SAP identifiers. These values are listed in Constants and Common Types later in this chapter. (Type INTERNET_ADDRESS.) See appendix B for a description of INTERNET_ADDRESS.	

(Continued)

Table 16-4. Data Indication Record (INTERNET_IND_IF) *(Continued)*

Field	Content
control	This is a packed record. (CONTROL_BYTES.) **Field** / **Content** hop_count — This value indicates the number of intermediate systems the message would have to be relayed through before it reaches the final destination. (0..OFF (16).) packet_kind — This field indicates the protocol identifier to be used in the header of the 3B PDU. It is supplied by the user sending the data and is delivered to the user at the destination system. A list of known values for packet types for Internet requests and indications is given in Constants and Common Types later in this chapter. (Type PACKET_TYPE.)
user_id	Pointer to a cell containing the user identifier which was supplied when the destination user's SAP was opened. (^CELL.)
data	Pointer to the data. The maximum data length is 1466 bytes. (Type BUF_PTR.)

CLOSE_INTERNET_SAP

This request allows an Internet layer user to close an Internet SAP.

Comdeck **B3XREQI**

Format **CLOSE_INTERNET_SAP (sap_id, user_id, return_code)**

Input **sap_id: sap_id_type**

Identifies the SAP that is to be closed.

user_id: ^cell

The user identifier that was passed by the user when the SAP was opened.

Output **return_code: close_internet_sap_status**

The status indication for the processed request. Following are status messages returned for the CLOSE_INTERNET_SAP procedure. For explanations, see Constants and Common Types later in this chapter.

```
close_sap_successful
sap_already_closed
mismatch_userid
```

Constants and Common Types

This section lists all the constants and common types used by the procedures described in this chapter. Each group of constants and common types is preceded by the name of the common deck in which they are defined.

Constants

Common deck B3DERRD

```
ier3bpdu_size = 42
```

Common deck B3DIERC

```
ier_unspecified_at_destination = 0,
ier_checksum_at_destination = 1,
ier_sap_closed = 2,
ier_resource_limit_at_dest = 3,
ier_unspecified_before_dest = 200(16),
ier_checksum_before_destination =  201(16),
ier_destination_unreachable =  202(16),
ier_hop_count =  203(16),
ier_packet_too_large =  204(16),
ier_not_echo_packet = 1001(16),
ier_not_echo_req = 1002(16)
```

Common deck B3DDFAU

```
max_data_length = 1466,
default_network_id = 0,
default_system_id_upper = 0,
default_system_id_lower = 0,
default_sap_id = 0
```

Common deck B3DDSAP

Interface via Transport (4) layer

```
directory_me_xp_sapid = 1020(10),
file_access_me_xp_sapid = 1021(10),
command_me_xp_sapid = 1022(10),
dep_log_me_xp_sapid = 1023(10),
dep_alarm_me_xp_sapid = 1024(10),
ind_file_access_me_xp_sapid = 1025(10),
ind_command_me_xp_sapid = 1026(10),
ind_log_me_xp_sapid = 1027(10),
ind_alarm_me_xp_sapid =1028(10)
```

Ephemeral Internet SAPs

```
request_ephemeral_sap = 0(10),
lowest_ephemeral_sap = 3001(10),
highest_ephemeral_sap = 0ffff(16)
```

Interface via Internet (3B) layer

```
routing_me_sapid = 1(10),
echo_me_sapid = 2(10),
error_me_sapid = 3(10),
directory_me_sapid = 20(10),
file_access_me_sapid = 21(10),
command_me_sapid = 22(10),
dep_log_me_sapid = 23(10),
dep_alarm_me_sapid = 24(10),
ind_file_access_me_sapid = 25(10),
ind_command_me_sapid = 26(10),
ind_log_me_sapid = 27(10),
ind_alarm_me_sapid = 28(10),
highest_dedicated_sap = 3000(10)
```

Common deck B3DPCKT

```
unknown_packet_type = 0,
xerox_routing_info_packet = 1,
xerox_echo_packet = 2,
xerox_error_packet = 3,
xerox_packet_exchange = 4,
xerox_sequenced_packet = 5,
experimental_packet = 16,
cdna_routing_info_packet = 17,
cdna_directory_packet = 18,
cdna_command_packet = 19,
cdna_log_packet = 20
```

Common Types

Common deck B3DCSAP

```
close_internet_sap_status =
  close_sap_successful,          The request to close SAP was
                                 successful.
  sap_already_closed,            The specified SAP is already
                                 closed.
  mismatch_userid)               The USERID does not match the
                                 SAP identifier that was passed
                                 when the SAP was opened.
```

Common deck B3DCOBY

```
control_bytes = packed record
  hop_count: 0 .. 0ff(16),
  packet_kind: packet_type,
recend
```

Common deck B3DECHT

```
echo_operation = (echo_null, echo_request, echo_reply)
```

Common deck B3DERRD

```
error_me_data = record
 error_number: internet_error_codes,
 error_parameter: 0 .. 0ffff(16),
recend
```

Common deck B3DFCIS

```
force_close_if = ^procedure (
      sap_id: sap_id_type;
      user_id: ^cell)
```

Common deck B3DERRD

```
internet_error_codes = 0 .. 0ffff(16)
```

Common deck B3DRTNT

`internet_return_codes =`	
`internet_success,`	The request was processed successfully.
`ineterror_nil_param,`	A NIL value was passed for the REQ_PARAM parameter on the Data Request procedure.
`ineterror_sosap,`	The source SAP that was specified is not in the SAP table.
`ineterror_dssap,`	The destination SAP that was specified is not in the SAP range.
`ineterror_data)`	No data or data that exceeds the maximum limit was passed through this request.

Common deck B3DOSAP

`open_internet_sap_status =`	
`open_sap_successful,`	The request to open the SAP was successful.
`illegal_dedicated_sap,`	The dedicated SAP is not in the expected range.
`nil_parameter_pointer,`	A NIL value was passed for the INPUT_PARAM or OUTPUT_PARAM parameters on the OPEN_INTERNET_SAP request.
`no_destination_proc,`	Destination procedure was not provided.
`sap_already_opened,`	Attempting to open a dedicated SAP that is already open.
`no_sap_entries_available,`	All SAPs are currently allocated.
`sap_3b_insuf_resorc,`	Insufficient memory to create a SAP table entry.
`internet_down)`	Internet services are not available.

Common deck B3DPCKT

```
packet_type = 0 .. 0ff(16)
```

Intranet Layer 17

Intranet Layer 17

This chapter discusses:

- An overview of the Intranet layer (layer 3A).
- The services required by the Intranet layer.
- The services provided by the Intranet layer.
- Constants and common types used in the Intranet layer's service requests.

Overview

The Intranet layer (3A) is a lower layer software component. It provides a generic interface between the higher layer CDCNET software and the individual network solutions' software (layers 1 and 2). Layers above Intranet and various management entities communicate with individual stream service routines (SSRs) through the Intranet layer. SSRs are layer 2 software.

Intranet layer services can be functionally divided into three groups:

- SAP management services.
- Data transfer services.
- Status indication services.

The SAP management services include:

Opening an Intranet layer SAP.

Closing an Intranet layer SAP.

The data transfer services include:

Transferring data to and from network solutions.

The status indication services include:

Sending 3A status update indications

User Interfaces to the Intranet Layer

Figure 17-1 illustrates the interfaces between the Intranet layer and its users. It also shows the interfaces between the Intranet layer and the lower layer which provides services to the Intranet layer.

All interfaces are invoked through procedural calls. The formats and definitions of the service requests from users and the indications sent by the Intranet layer are described in Services Provided, later in this chapter.

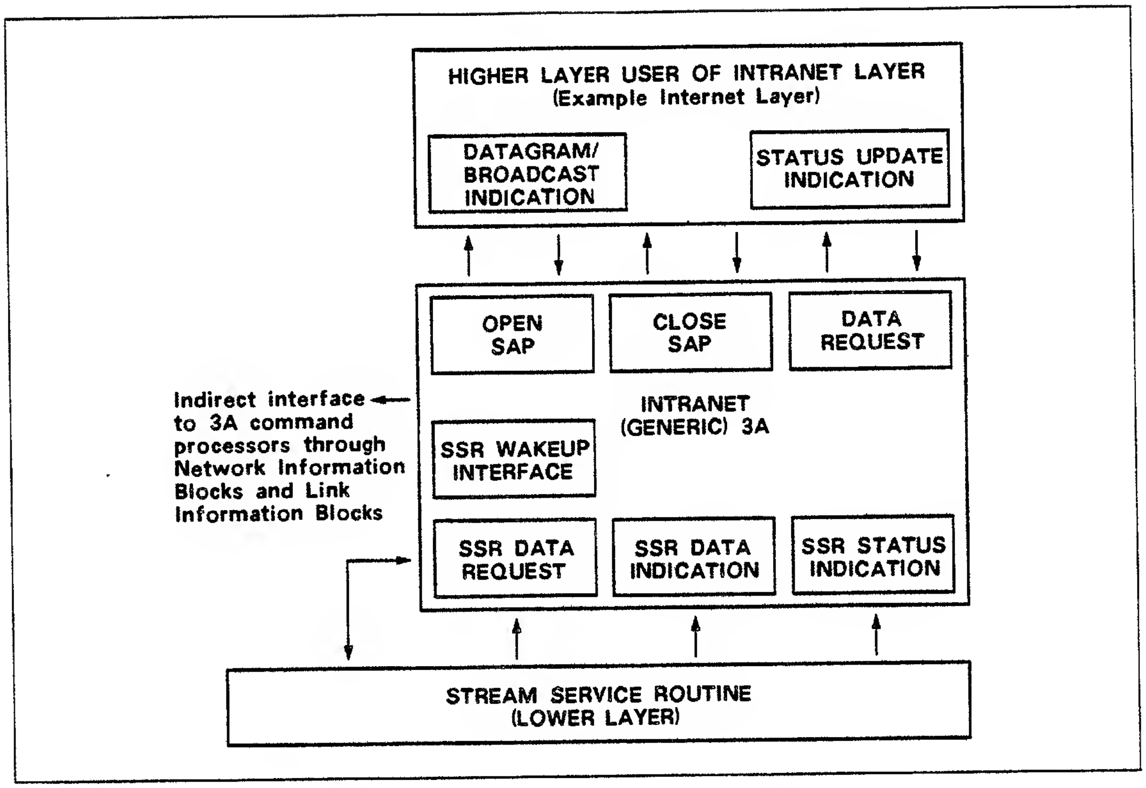


Figure 17-1. Intranet Layer Overview

Opening an Intranet Layer SAP

As mentioned earlier, users communicate with individual network solutions through the Intranet layer. Before any data can be sent or received, the users must open an Intranet SAP.

Each user opens its own SAP by calling the OPEN_3A_SAP procedure. In this procedure, the user passes the following information:

- The network identifier of the network solution on which the user wishes to send and receive data. A value of zero for this field indicates that the user wants to send or receive data on all network solutions.
- The protocol type of the user. The Intranet layer needs the protocol type to identify the user so that incoming messages can be routed to it.
- The addresses of two procedures, DATA_IND_PROC and STATUS_IND_PROC, which are used by the Intranet layer to send data and status indications.

The Intranet layer in turn, returns the following:

- The addresses of two procedures, CLOSE_3A_SAP and DATA_REQUEST_3A, which are called by the user when it wishes to close the SAP and when it wishes to send data, respectively.
- The SAP identifier that uniquely identifies the user with the opened SAP.
- Status indication of the OPEN_3A_SAP request.

Sending Data

When the user has data to be sent downline, it calls the DATA_REQUEST_3A procedure. In this procedure, the user specifies the following:

- The identifier of the network solution on which the data should be transmitted.
- The system identifier of the destination system.
- The user's SAP identifier that was assigned by the Intranet layer when the Intranet SAP was opened.
- Address of the buffers containing the user data.

Intranet then builds the 3A header and does the following:

- Places the associated header information for the type of network solution specified in front of the data.
- Enqueues the datagram in the associated 3A queue.
- Changes the network status if the network gets congested.
- Informs all the users of any status changes.
- Sends an intertask message to the SSR if it is waiting for a message.

  For more information on SSRs, see chapters 18, 19, and 20 in this manual. Also see the SSR_SLEEPING field in the LIB which is described later in this section; the LIB record is given in Constants and Common Types, later in this chapter.

Receiving Data

When a datagram is received on a network solution and is transmitted upline, the Intranet layer does the following:

1. Strips off the 3A header.
2. Gets the protocol type that is specified in the header. The protocol type indicates the user for whom the datagram is intended. (See Constants and Common Types later in this chapter for more information on headers and protocol types.)
3. Retrieves the address of the DATA_IND_PROC (which was supplied by the user when it opened the Intranet SAP) from the Intranet SAP table.
4. Sends the datagram upline.

Network Status

When the Intranet layer receives information on the status of a network solution, it does the following:

- Updates the status fields in the network information and the link information blocks for the associated network solution.
- Notifies all the users who have opened the Intranet SAPs to the specified network solution through the STATUS_IND_PROC procedure, which was supplied by the user when the SAP was opened.

Intranet Headers

The Intranet header is identical to the IEEE 802.2 link layer header. The Intranet layer uses this common header for all network solutions supported by CDCNET.

The information in the header is used to route the incoming data units to the appropriate Intranet SAP. The contents of this header are based on the users of 3A in the source and destination system.

The Intranet layer is responsible for adding and removing the 3A headers. Headers are added by the GENERIC_3A_DATA procedure. An appropriate header type is selected based on what is specified in the NETWORK_TYPE field in the network information block (NIB). The header is removed before a data indication is sent to the Intranet layer users.

For more information on SSRs, see chapters 18, 19, and 20 in this manual. Also see Constants and Common Types in this chapter for CYBIL definitions of the headers and header types.

Intranet Layer Data Structures

The Intranet layer maintains and uses two main data structures:

- The network information block (NIB).
- The link information block (LIB).

As seen in figure 17-2, NIB is the interface between the Intranet layer and its higher layer users. LIB is the interface between the Intranet layer and the SSRs.

Following is a brief description of these data structures. The CYBIL record type definitions for these data structures are given in Constants and Common Types, later in this chapter.

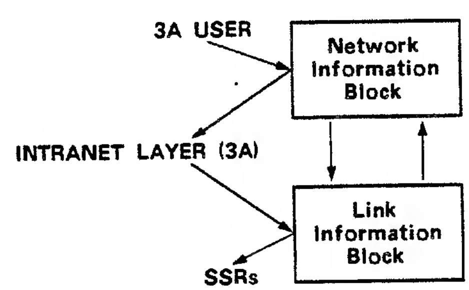


Figure 17-2. Intranet Layer, NIB, LIB Relationship

Network Information Block (NIB)

A NIB is a table that exists for each configured network solution. This table contains network-related information such as:

- The network identifier.
- The cost of the network solution.

A NIB is created when a command to define a network is executed by the 3A command processors. The command parameters can be used to provide the values of different fields in the NIB. (For more information on network definition commands, see CDCNET Network Configuration and Site Administration Guide).

All NIBs for a particular DI are linked together through a data structure called the network solution list. This data structure helps Intranet and its command processors search for a particular NIB. Each NIB is also linked together to its associated LIB. (The LIB is described next.)

When a command to cancel a particular network is executed, the corresponding NIB and the pointer to this NIB block from any associated LIB are deleted.

Link Information Block (LIB)

A LIB is a table that exists for each configured trunk. This table contains link-related information needed to provide:

- The services and functions associated with a particular link.
- An interface between the Intranet layer and the SSRs.

A LIB is created when a command to define a trunk is executed.

The LIB contains:

- Physical layer (1) information.
- Link-related information.
- Intranet layer-related information.

As in the case of NIBs, all LIBs for a particular DI are linked together through the network solution list. This list is used to search for a particular trunk by the 3A command processor.

When a command to cancel a particular trunk is executed, the corresponding LIB is deleted.

Services Required

For the Intranet layer to provide services to its own users it, in turn, depends on the services of other software components. Brief descriptions of the services of each of these software components follow.

Data Link Layer Interfaces

The following Data Link layer interfaces are used by the Intranet layer.

SSR_DATA_REQUEST Procedure

The SSR_DATA_REQUEST procedure is supplied by the Intranet layer to do the following:

- Allow an SSR to retrieve a datagram from the 3A output queue.
- Transmit the next data unit on a specified network solution.

This service is initiated when the SSR calls this procedure to get the next data unit to be transmitted.

If a data unit is not available for transmission, the Intranet layer sets the SSR_SLEEPING field in the LIB to TRUE.

When data becomes available for transmission, the Intranet layer does the following:

- Sets the SSR_SLEEPING field in the LIB to FALSE.
- Sends the SSR an intertask message informing it that data needs to be transmitted.

The SSR knows the address of the SSR_DATA_REQUEST procedure through the SSR_DATA_REQ_PROC field in the LIB.

SSR_DATA_INDICATION Procedure

The SSR_DATA_INDICATION procedure is supplied by the Intranet layer to allow an SSR to indicate that a datagram has been received on the network solution and needs to be transmitted upline.

The SSR knows the address of this procedure through the SSR_DATA_INDIC_PROC field in the LIB.

SSR_STATUS_INDICATION Procedure

The SSR_STATUS_INDICATION procedure is supplied by the Intranet layer to allow an SSR to indicate the status of the network solution.

The SSR knows the address of this procedure through the SSR_STATUS_INDIC field in the LIB.

Services Provided

This section defines the external services provided by the Intranet layer to its users. It describes the major data structures used, and the external interfaces between the Intranet layer and its users.

Common Types used by the Intranet layer and detailed explanations of returned status messages are given in Constants and Common Types at the end of this chapter.

User Interface

As mentioned earlier, the user has to open an Intranet SAP before any other service can be requested.

The OPEN_3A_SAP is an externally referenced (XREF) procedure which is contained in the common deck A3XUPL. Intranet layer users include this deck in their calling modules. The common deck A3XUPL contains calls to several decks. One of these decks, A3DPRCS, contains definitions of the datagram and status procedures that are returned by the Intranet layer as output parameters in an OPEN_3A_SAP request.

The following pages describe the Intranet procedures that provide various services to the users.

OPEN_3A_SAP

This request allows the Intranet user to open a 3A SAP. This procedure is called to open SAPs to all available network solutions. Through this procedure, the user specifies its protocol type and provides addresses of two user procedures. These procedures are called by Intranet to provide data- and status-related indications.

Comdeck **A3XUPL**

Format **OPEN_3A_SAP (protocol_type, data_ind_proc, status_ind_proc, network_id, close_3a_sap_proc, data_request_3a_proc, sap, open_status)**

Input

protocol_type: protocol_range_type

The protocol type to be associated with the SAP. For details, see Constants and Common Types later in this chapter.

data_ind_proc: user_datagram_proc_type

A pointer to the user procedure which is called by the Intranet layer to send datagrams upline.

status_ind_proc: user_status_proc_type

A pointer to the user procedure which is called by the Intranet layer to send status indications.

network_id: network_id_type

The identifier of the network whose SAP is being opened. A network_id equal to zero indicates that the SAP is open for all network solutions. A network_id not equal to zero indicates that the SAP is open for a specific network solution.

Output

close_3a_sap_proc: close_3a_sap_proc_type

A pointer to the procedure which is called by the user when it needs to close this SAP.

data_request_3a_proc: data_request_3a_proc_type

A pointer to the procedure which is called by the user to send a datagram downline.

sap: intranet_sap_type

The SAP identifier specified by the Intranet layer. It must be specified by the user when it wants to close this SAP.

open_status: |3a_status_type

This is the status indication for the processed request. Following is a list of status messages applicable to this request. For explanations, see Constants and Common Types later in this chapter.

```
request_processed
sap_out_of_range
sap_active
sap_not_active
```

CLOSE_3A_SAP

This request allows the Intranet layer user to close a currently open Intranet layer SAP. The address for this procedure is obtained by the user when it opens a 3A SAP.

Comdeck **A3DPRCS**

Format **CLOSE_3A_SAP (sap, close_status)**

Input **sap: intranet_sap_type**

The SAP identifier that was specified by the Intranet layer when this SAP was opened.

Output **close_status: l3a_status_type**

This is the status indication for the processed request. Following is a list of status messages applicable to this request. For explanations, see Constants and Common Types later in this chapter.

```
request_processed
sap_out_of_range
sap_active
sap_not_active
```

DATA_REQUEST_3A

This request allows the Intranet user to send datagrams on a specific network solution. The user must provide the destination address and the SAP identifier that was passed in the open SAP procedure.

Comdeck A3DPRCS

Format **DATA_REQUEST_3A (network_id, destination_address, sap, data_ptr, request_processed)**

Input

network_id: network_id_type

The network identifier of the network solution.

destination_address: system_id_type

The system address of the destination system.

sap: intranet_sap_type

The SAP identifier that was returned by the Intranet layer when the SAP was opened.

data_ptr: buf_ptr

A pointer to a chain of buffers containing the user data.

Output

request_processed: boolean

Indicates the status of the request.

TRUE	The request was processed.
FALSE	The request was not processed.

Indication Services

Indication services provide the means by which the Intranet layer informs the user that a certain event has occurred.

The following two indication routines are provided by the user when an Intranet SAP is opened:

- USER_DATA_IND_PROC indicates to the user that peer across the network has data for it.
- USER_STATUS_IND_PROC indicates the status of a network solution.

Data Indication Procedure

The Intranet layer informs the user that there is data for it by calling the USER_DATA_IND_PROC. The address of this procedure is passed to the Intranet layer when a user opens an Intranet SAP. The indication that is passed in the procedural call is defined below:

```
TYPE
user_datagram_proc_type = ^procedure (
multicast: boolean;
receiving_network_id: network_id_type;
originating_system_id: system_id_type;
VAR data_ptr: buf_ptr);
```

multicast: boolean

This parameter specifies if the data was received through a datagram or a broadcast service.

TRUE	Data was received through a broadcast service.
FALSE	Data was received through a datagram service.

receiving_network_id: network_id_type

The network identifier of the network solution receiving the data unit. See appendix B for a description of NETWORK_ID_TYPE.

originating_system_id: system_id_type

The system identifier of the source system sending the data unit. See appendix B for a description of SYSTEM_ID_TYPE.

data_ptr: buf_ptr

A pointer to the record that contains the message.

Status Update Indication

The Intranet layer gives the status of a requested network solution to the user by calling the USER_STATUS_IND_PROC procedure. The address of this procedure is passed to the Intranet layer when a user opens an Intranet SAP. Intranet layer users check the new status in the NIB whose address is passed as a parameter in the procedure. The following is the indication that is passed in the procedural call:

```
TYPE
    user_status_proc_type = ^procedure(
        network_nib: ^nib_type);
```

network_nib: ^nib_type

The pointer to the NIB. NIB_TYPE is a record that consists of several fields. The following fields in the NIB are useful to the Intranet layer user. The NIB record is fully described in Constants and Common Types later in this chapter.

network_id: network_id_type

The network identifier of the network solution whose status is being reported.

oetwork_status: network_status_type

The status of the network solution.

max_data_unit_size: integer

The maximum length (0 through 65535) of the data unit that is allowed over this network solution.

multicast_network: boolean

This field indicates whether or not this network solution supports a multicast service.

TRUE	The network supports a multicast service.
FALSE	The network does not support a multicast service.

Constants and Common Types

This section lists the constants and common types used in the Intranet layer interfaces. It also lists and explains status messages returned for the processed requests. Each group of types or constants are preceded by the name of the common deck in which it is defined.

Constants

Common deck A3DPROT

```
all_3a_networks = 0

esci_message_size = 46

init_me_3a_prtcl = 16,
dod_internet_prtcl = 8,
unknown_prtcl = 3,
diag_me_3a_prtcl = 2,
routing_me_3a_prtcl = 12,
xerox_internet_3a_prtcl = 4

max_message_size = 1497

max_sap_table_entry = 4

hdlc_3a_header_size = 3
mci_3a_header_size = 17
mci_l2_header_size = 14
esci_3a_header_size = 17
esci_l2_header_size = 14
```

Common Types

Common deck CMDLIB

```
13_lib_type = record
  nib_ptr:  ^cell,
  output_qcb:  qcb@,
  nxt_lib_ptr:  ^13_lib_type,
  link_status:  link_status_type,
  ssr_task_id:  taskid,
  ssr_tracing,
  ssr_collecting_stats,
  ssr_sleeping_lock,
  ssr_sleeping:  boolean,
  ssr_data_req_proc:  ssr_data_req_proc_type,
  ssr_data_ind_proc:  ssr_data_ind_proc_type,
  ssr_status_ind_proc:  ssr_status_ind_proc_type,
  link_type:  owner_type,
  trunk_name:  clt$name,
  next_linked_lib:  ^cell,
  lib_defined:  boolean,
recend
```

Common deck A3DPROT

`13a_status_type = (request_processed,`	The request was processed.
`sap_out_of_range,`	Invalid protocol type.
`sap_active,`	SAP has already been opened for this protocol type.
`sap_not_active)`	SAP was previously opened for this protocol type.

Common deck A3DPRCS

```
close_3a_sap_proc_type = ^procedure (
    sap: intranet_sap_type;
VAR close_status: l3a_status_type)

protocol_range_type = 0 .. 0ff(16)

user_datagram_proc_type = ^procedure (
  multicast: boolean;
  receiving_network_id: network_id_type;
  originating_system_id: system_id_type;
  VAR data_ptr: buf_ptr)

user_status_proc_type = ^procedure (
  network_nib: ^nib_type)

data_request_3a_proc_type = ^procedure (
    network_id: network_id_type;
    destination_address: system_id_type;
    sap: intranet_sap_type;
VAR data_ptr: buf_ptr;
VAR request_processed: boolean)

intranet_sap_type   = 0 .. 65535
```

Common deck CMDLIB

```
network_range_type =
  hdlc_network,
  esci_network,
  mci_network,
  x25_network)

network_status_type = net_up,
  net_inactive,
  net_congested,
  net_terminate)

nib_type = record
next_nib:  ^nib_type,
network_type:  network_range_type,
network_status:  network_status_type,
network_id:  network_id_type,
network_name:  clt$name,
network_cost:  0 .. 0ffff (16),
relay_allowed:  boolean,
multicast_network:  boolean,
cdna_routing_info_nw:  boolean,
rotary:  boolean,
cdna_xerox_broadcast_addr:  system_id_type,
max_data_unit_size:  0 .. 0ffff(16),
intranet_header_size:  0 .. 0ffff(16),
congestion_threshold:  0..255,
un_congestion_threshold:  0..255,
lib_ptr:  ^cell,
intranet_sds_data1:  intranet_sds_expanded_data,
intranet_sds_data2:  intranet_sds_expanded_data,
intranet_sds_data_ptr:  ^intranet_sds_expanded_data,
recend
```

(Continued)

The Data Link layer provides the services to enable communication over four types of network solutions:

- A CYBER 170-based network.
- A CYBER 180 CDNA network.
- An Ethernet-based network.
- A high-speed data link channel.

The Data Link layer consists of three groups of software routines, each group supporting network connection to a specific type of network solution. These software routine groups include the following:

- Mainframe channel interface (MCI) stream service routine (SSR) supporting a CYBER 170/180 network solution.
- Ethernet serial channel interface (ESCI) SSR supporting connection to bit-synchronous, point-to-point communication lines (an Ethernet network solution).
- High-speed data link channel (HDLC) SSR supporting an HDLC network solution.

The software routine groups are discussed in subsequent chapters of part III of this manual.

Data Link Layer Data Structure

The Data Link layer uses one data structure, the link information block (LIB). The LIB provides the interface between the Data Link layer and the Network layer. The LIB provides the physical characteristics, the link-related characteristics, and the Network-related characteristics of a specific trunk.

Link Information Block

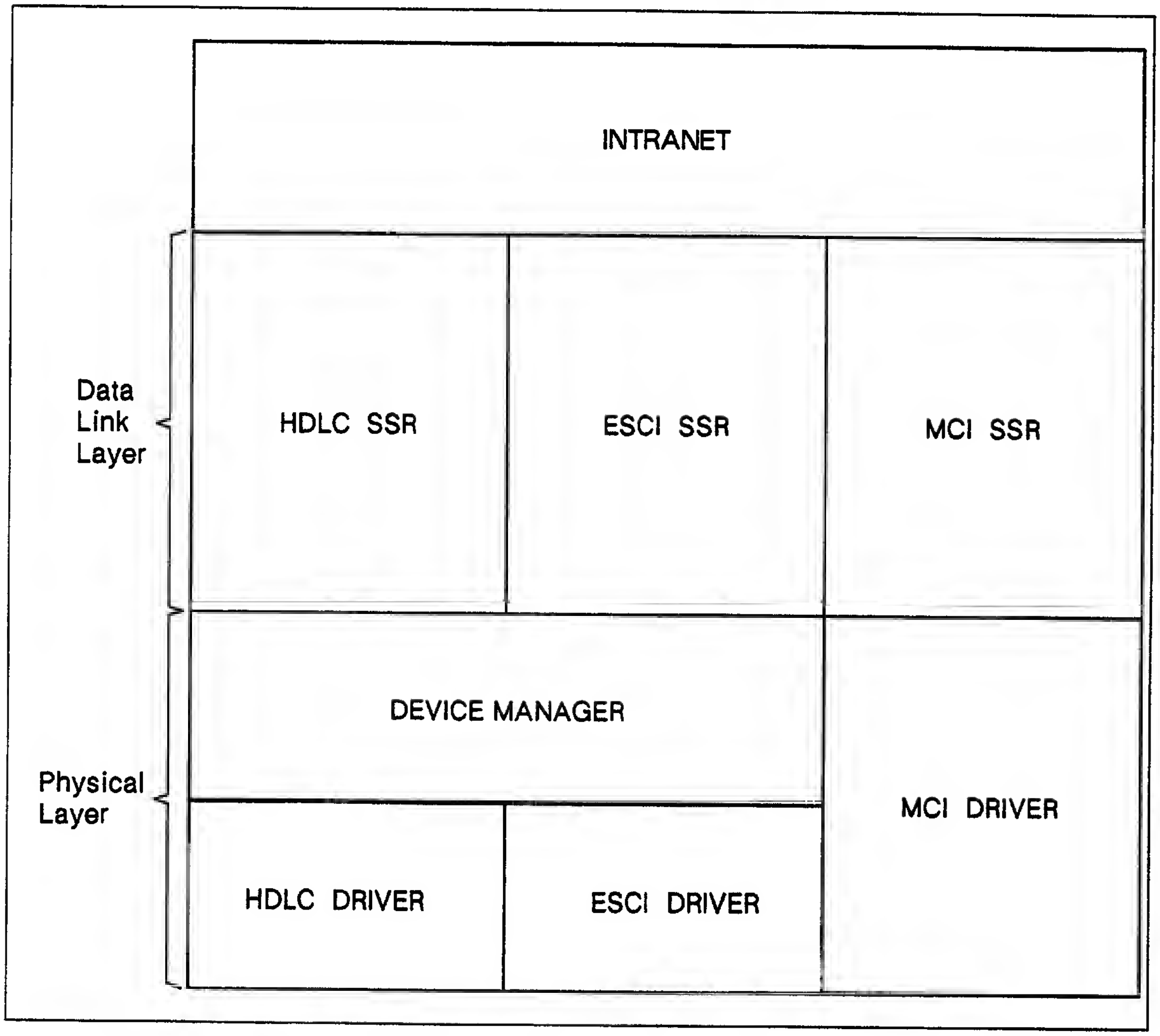
The LIB is created during configuration. During configuration, a command processor configures the subsystem (the SSR, the driver, and the physical layer) to the network. A command processor creates a LIB for each trunk. Each LIB contains pointers to the procedures with which the Data Link layer provides services to the Network layer, as well as the trunk characteristics.

The Constants and Common Types section of each chapter contains the LIB definition for each particular type of network trunk.

(Continued)

Data Link Layer

The Data Link layer is a lower layer software component that provides an interface between the Network layer and the Physical layer software. The Data Link layer and the Physical layer software together provide access to a communication medium. An overview of the Data Link layer is shown in the following figure.



Data Link Layer Overview

The Data Link layer provides the services needed to transfer data to and from a particular network. Specifically, the Data Link layer provides the following services:

- Retrieves a data unit from the 3A output queue of a specific network.
- Transmits a data unit to a specific network.
- Provides data and status indication services.

(Continued on other side)

(Continued)

Services Required

The Data Link layer depends upon the services of the Physical layer software (the appropriate Driver) to establish, maintain, and disconnect the physical link between the communication medium and CDCNET. Each implementation of the Data Link layer requests these services from the lower layer slightly differently.

The services required by each implementation of the Data Link layer are presented in the appropriate chapter discussing the related SSR.

Physical Layer Interfaces

The Data Link layer communicates with the Physical layer through the physical layer software (driver). The Data Link layer issues a direct call to the appropriate driver to request a service. The called driver processes the direct call and issues an interrupt to the hardware. This interrupt requests the service originally requested by the Data Link layer. The hardware processes the interrupt and returns an interrupt, reporting the success or lack of success in processing the interrupt, to the Physical layer software.

User Interface

The user interface between the Data Link layer and the Network layer is the LIB. There is more information about the LIB earlier in this overview.

Services Provided

The Data Link layer provides data request services and indication services to the Network layer. The Data Link layer provides these services through the LIB. These services include the transfer of data from the Network layer to the Physical layer (data request services), the transfer of data from the Physical layer to the Network layer (data indication service), and the transfer of link status information from the Physical layer to the network layer (status indication service). The Data Link layer also provides interface startup and termination services to its users by responding to intertask messages (ITMs) from its users.

The Data Link layer provides the following services to its users:

- Data request service
- Indication service

The data request service allows the Data Link layer to transfer data from the network layer 3A output queue to the Physical layer. The Data Link layer provides the service when the layer detects that data are waiting for transfer from the Network layer 3A output queue to the medium.

(Continued)

(Continued)

The data request service also allows the network layer to request the transfer of data from the network layer 3A output queue to the Physical layer. The network layer requests activation of the Data Link layer task when the Network layer output queue has information available for transfer and detects that the Data Link layer task is currently not active. Only the MCI SSR and the ESCI SSR provide this data request service.

The indication service allows the Data Link layer to inform the Network layer that data or status information is available for transfer to the network layer from the Physical layer.

The Data Link layer provides these services by issuing direct calls to the appropriate network layer procedure. The addresses of the following procedures are maintained in the LIB:

- Data request procedure (SSR_DATA_REQ_PROC)
- Data indication procedure (SSR_DATA_INDICATION_PROC)
- Status indication procedure (SSR_STATUS_IND_PROC)

The Data Link layer accepts from the Network layer data waiting for transfer to the Physical layer. The Data Link layer provides this service by issuing a direct call to the Network layer procedure SSR_DATA_REQ_PROC. This procedure returns to the Data Link layer the location of the next data waiting transfer to the Physical layer.

The Data Link layer transfers to the Network layer the data received from the Physical layer. The Data Link layer provides this service by issuing a direct call to the Network layer procedure SSR_DATA_INDICATION_PROC. This Data Link layer passes to this procedure the location of the next data recieved from the Physical layer.

The Data Link layer transfers to the network layer the status information received from the Physical layer. The Data Link layer provides this service by issuing a direct call to the Network layer procedure SSR_STATUS_IND_PROC. The SSR passes to this procedure the location of the status information received from the Physical layer.

Mainframe Channel Interface Stream Service Routine 18

Mainframe Channel Interface Stream Service Routine 18

This chapter discusses the mainframe channel interface (MCI) stream service routine (SSR). Topics covered are:

- An overview of the MCI SSR, including the MCI SSR user interface, initialization of the MCI SSR, and the flow of data through the MCI SSR.
- The services required by the MCI SSR.
- The services provided by the MCI SSR to its users.
- A list of the CYBIL constants and common types used by the MCI SSR.

Overview

The MCI SSR is a software component that supports connection to a NOS or NOS/VE network solution. The MCI SSR executes as a task on the main processor board (MPB) of the DI and supports connection to the following Control Data products:

- NOS Network Products
- NOS/VE CDNA network solutions

A different 3A layer software component uses the MCI services for each of these products. The Block Protocol Interface Program (BIP) and the Service Module (SVM) are the users of the MCI services in a NOS Network Product. Intranet is the user of the MCI services in a NOS/VE CDNA Network Solution product.

The MCI SSR provides the same interfaces to all of the 3A layer users. This chapter refers to the user of the MCI stream service routine as the 3A layer.

MCI SCR User Interfaces

The MCI SSR, together with the MCI Driver, comprise the MCI subsystem. This MCI subsystem provides the software interface between the 3A layer and the MCI driver. Figure 18-1 shows the interfaces between the MCI SSR and its users, and the interfaces between the MCI SSR and the MCI driver that provides services required by the MCI SSR. The MCI SSR and the 3A layer communicate through direct calls and intertask messages. The MCI SSR and the MCI driver communicate through intertask messages (ITMs) and interrupts.

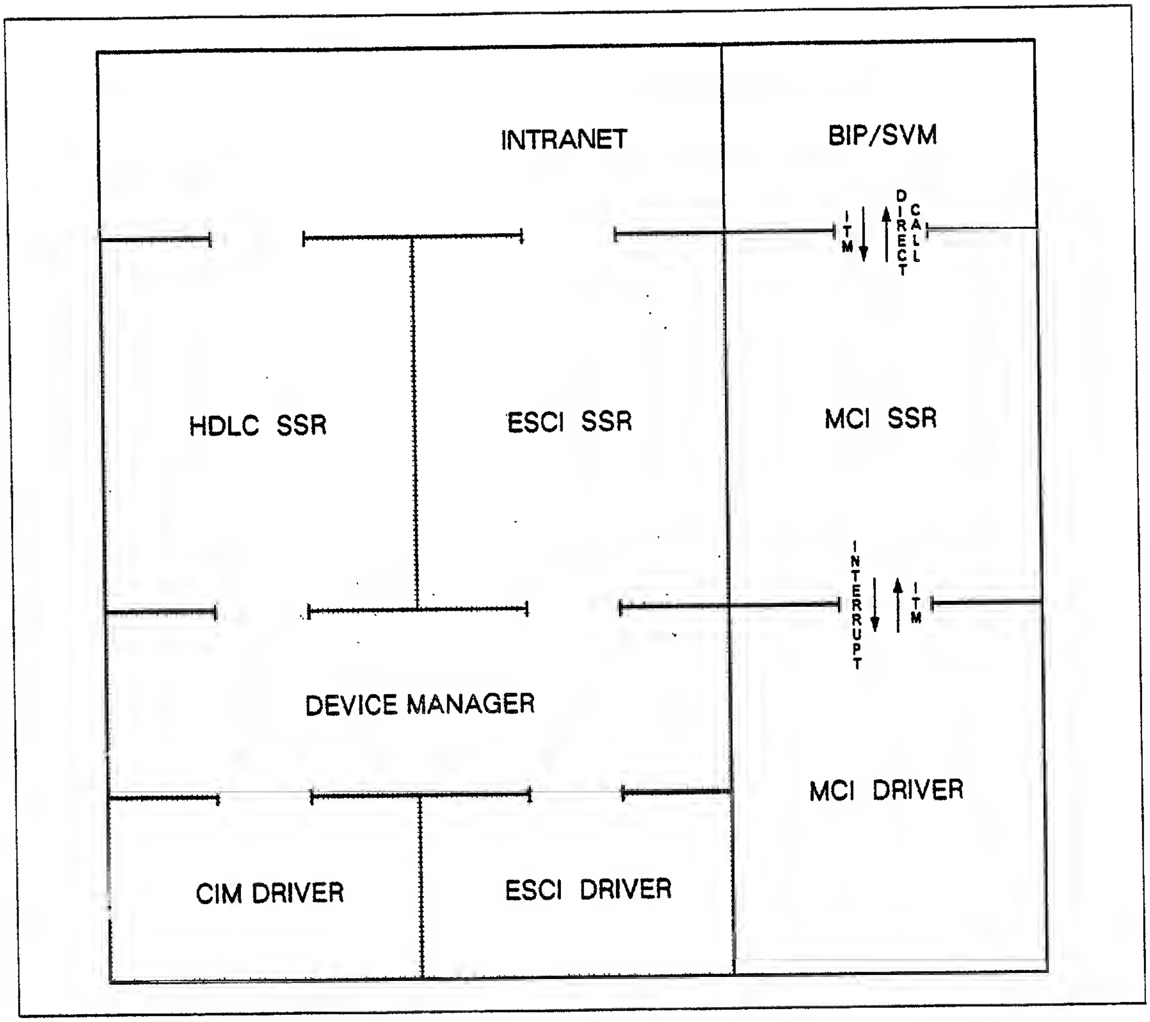


Figure 18-1. MCI SSR Interfaces

This chapter refers to data units transferred over an MCI channel as datagrams.

MCI Initialization

The MCI SSR executes as a task and receives an ITM from either the MCI boot startup procedure, a command processer, or its 3A layer user. The MCI boot startup procedure (or a command processor) creates and initializes the link information block (LIB), which configures the MCI subsystem (MCI SSR, MCI Driver, and MCI channel) to the network. The network information block (NIB) is also created and initialized. The MCI SSR starts executing as a task. The MCI SSR executes its initialization procedure and puts itself into an idle state, waiting for a startup ITM.

The MCI boot startup procedure, command processor, or the 3A layer sends an ITM to the MCI SSR. After initialization and startup completes, the MCI SSR returns an ITM reporting the success or failure of the startup ITM.

The 3A layer user issues an ITM to the MCI task whenever the 3A layer output queue has datagrams available for transfer. The 3A layer issues the ITM only if the 3A layer detected that the MCI SSR previously emptied the 3A output queue.

MCI Channel Header

The datagrams passed to the MCI SSR from the 3A layer have a channel header, containing general status information. The MCI SSR gets the lower 6 bits of the channel header and places those bits into the lower 6 bits of the general status frame. The mainframe peripheral processor (PP) requests general status to determine the status of the MCI subsystem.

MCI SSR Data Structure

The MCI SSR and the 3A layer share a common data structure, the link information block (LIB). This section discusses the LIB as used by the MCI SSR.

Link Information Block

The LIB contains the configuration parameters controlling MCI subsystem operation. The MCI LIB consists of:

- Link layer (L2) section that provides the information needed for the MCI SSR to control operation of the link.
- Network layer (L3) section that provides the information needed for maintaining the output data queue to the 3A layer, the 3A layer interface procedures, and other configuration parameters.

Constants and Common Types, later in this chapter, contains a CYBIL definition of the LIB record.

Services Required

The MCI SSR depends upon the interrupt-level services of the MCI Driver to establish, maintain, and disconnect the physical link between the mainframe and CDCNET. The MCI SSR requires the following services from the MCI Driver:

- Subsystem startup
- Subsystem shutdown
- Output data available
- Input buffer available
- Release data buffers

The MCI SSR issues a trap interrupt to the MCI Driver, requesting the required service. The issued interrupt traps into the interrupt vector table to the MCI Driver. The MCI Driver issues an intertask message to the MCI SSR upon completion of the requested service. The MCI SSR then processes the intertask message.

Because the MCI Driver has no interfaces available to a gateway software writer, the MCI Driver will not be described in this manual.

Services Provided

This section describes the services provided by the MCI SSR to the 3A layer, and how the MCI SSR uses the MCI LIB data structure to provide these services. Constants and Common Types, later in this chapter, provides the CYBIL definition of the MCI LIB data structure. The services provided include the following:

- Data request services
- Indication services

The MCI SSR provides these services through the LIB. The MCI SSR uses information in the LIB to call the 3A layer data request and indication procedures.

The MCI SSR also provides services to the 3A layer through ITMs. These services were described previously in this chapter.

Data Request Services

The MCI SSR provides data request services to the 3A layer. These data request services include:

- Data request service
- Turnaround back quickly service

The data request service allows the 3A layer to transfer datagrams to the MCI channel. Normally, the MCI SSR issues a call to the 3A layer data request procedure through information in the LIB when the MCI SSR is ready for more output data.

The turnaround back quickly service allows the block interface program of the NOS interface software to transfer acknowledge datagrams to the MCI channel if the MCI SSR is already active.

When the 3A layer has output data available and detects that the SSR_SLEEPING field of the LIB record is TRUE, the MCI SSR is inactive. The 3A layer then issues an ITM (MCI_DATA_AVAILABLE) to the MCI SSR to activate the MCI SSR and ready the MCI SSR for datagram transfer. However, if the SSR_SLEEPING field of the LIB record is FALSE, the MCI SSR is already active.

Data Request Service

This service allows the 3A layer to transfer datagrams to the MCI channel when the datagrams are available in the 3A layer output queue.

Data Request Procedure

This procedure allows the MCI SSR to request datagrams from the 3A layer. The MCI SSR calls the SSR_DATA_REQ_PROC to request datagrams from the 3A layer output queue. The MCI SSR calls this procedure through the LIB. The MCI SSR then dequeues the data and passes the data to the MCI channel. The data request procedure call is defined below:

```
TYPE
    ssr_data_req_proc_type = ^procedure (
    p: ^cell;
    VAR q: buf_ptr
    ),
```

p: ^cell

The pointer to the LIB for the channel to which the data is destined.

q: buf_ptr

The pointer to the datagram received from the 3A layer.

Turnaround Back Quickly Service

This service is provided for BIP/SVM only. This service allows the 3A layer to transfer datagrams to the MCI channel quickly if the MCI SSR is already active. BIP requests this service by issuing a direct call to the MCI SSR procedure PROCESS_BACK.

Refer to chapter 21 of this manual for more information on BIP.

Indication Services

The MCI SSR indication services allow the MCI SSR to inform the 3A layer that data and status information is available for transfer to the 3A layer. The indication services facilitate the transfer of datagrams and status information. These indication services include:

- Data indication service
- Status indication service

Data Indication Service

The data indication service allows the Network layer to receive datagrams from the MCI channel.

Data Indication Procedure

This procedure allows the MCI SSR to indicate that datagrams have been received from the mainframe, and to transfer those datagrams to the 3A layer. The MCI SSR calls the SSR_DATA_IND_PROC through the procedure pointer in the MCI LIB. The data indication procedure call is defined below.

```
TYPE
    ssr_data_ind_proc_type = ^procedure(
    p: ^cell
    VAR q: buf_ptr
    ),
```

p: ^cell

The pointer to the LIB for the host computer channel from which the datagrams have been received.

q: buf_ptr

The pointer to the datagram.

Status Indication Service

This service allows the 3A layer to receive updated information about the status of the link from the MCI SSR.

Status Indication Procedure

This procedure allows the MCI SSR to indicate the status of the link to the 3A layer. The MCI SSR calls the SSR_STATUS_IND_PROC procedure through the procedure pointer in the LIB. The address of this procedure is passed to the 3A layer through the LIB. The status indication procedure call is defined below:

```
TYPE
    ssr_status_ind_proc_type = ^procedure (
    p: ^cell;
    s: link_status_type
    );
```

p: ^cell

The pointer to the LIB host computer channel.

s: link_status_type

This parameter indicates the current status of the link. The parameter has the following values:

Field Value	Meaning
link_uninitialized	Link has not been initialized and is not available for traffic.
link_initializing	Link has requested initialization data through the bootstrap data unit mechanism.
link_up	Link is available for user traffic.
link_idle	Link is physically available, but traffic is in the process of being suspended.
link_not_available	Link is logically not available, possibly in use by diagnostics.
link_processing_up	Link is in the process of starting up.
link_processing_idle	Link is in the process of being idled.
link_processing_term	Link is in the process of being terminated.
link_down	Link is physically not available.

Constants and Common Types

The MCI Stream Service Routine uses the following common types when providing services to its users.

Common Types

Common deck CMCLIBS

```
mci_lib_type = record
    13:                         13_lib_type,
    12:                         12_lib_type
recend
```

Common deck CMDLIB2

```
12_mci_lib_type = record
    card_slot:                  0 .. 7,
    stats_flag                  boolean,
    msg_length                  0 .. 2048,
    upline_message_timeout      2 .. 64,
    auto_queuing:               boolean,
    c170_np_addr:               ^cell
recend
```

Common deck CMDLIB3

```
13_lib_type = record
    nib_ptr:                    ^cell,
    output_qcb:                 qcb@,
    nxt_lib_ptr:                ^3b_lib_type,
    link_status:                link_status_type,
    ssr_task_id:                taskid
    ssr_tracing,
    ssr_collecting_stats,
    ssr_sleeping:               boolean,
    ssr_sleeping_lock,
    ssr_data_req_proc:          ssr_data_req_proc_type,
    ssr_data_ind_proc:          ssr_data_ind_proc_type,
    ssr_status_ind_proc:        ssr_status_ind_proc_type,
    link_type:                  owner_type,
    trunk_name:                 clt$name,
    next_linked_lib:            ^cell,
    lib_defined:                boolean
recend

link_status_type = (
    link_uninitialized,
    link_initializing,
    link_up,
    link_idle,
    link_not_available,
```

```
link_processing_up,
link_processing_idle,
link_processing_term,
link_down)
```

Ethernet Serial Channel Interface Stream Service Routine 19

Ethernet Serial Channel Interface Stream Service Routine 19

This chapter discusses the Ethernet Serial Channel Interface (ESCI) stream service routine (SSR). Topics covered are:

- An overview of the ESCI SSR, including the ESCI SSR user interface and initialization of the ESCI SSR.
- The services required by the ESCI SSR.
- The services provided by the ESCI SSR to its users.
- A list of the CYBIL constants and common types used by the ESCI SSR.

Overview

The ESCI SSR is a software component that supports connection to an Ethernet network solution. The ESCI SSR executes as a task on the main processor board (MPB) of the DI. The MPB interfaces with an intelligent peripheral, a microprocessor-driven peripheral card. The intelligent peripheral, supporting connection to an Ethernet network solution, is referred to as the ESCI in this chapter.

ESCI SSR User Interfaces

The ESCI SSR, together with the Device Manager and the ESCI Driver, comprise the ESCI subsystem. This ESCI subsystem provides the software interface between Intranet and the Ethernet trunk. Figure 19-1 shows the interfaces between the ESCI SSR and its users. The figure also shows the interfaces between the ESCI SSR and the ESCI Driver provided by the Device Manager. The ESCI SSR and the Intranet layer communicate through direct calls and intertask messages. The ESCI SSR and the Device Manager communicate through intertask messages and direct calls.

This chapter refers to the data units transferred over an ESCI interface as frames.

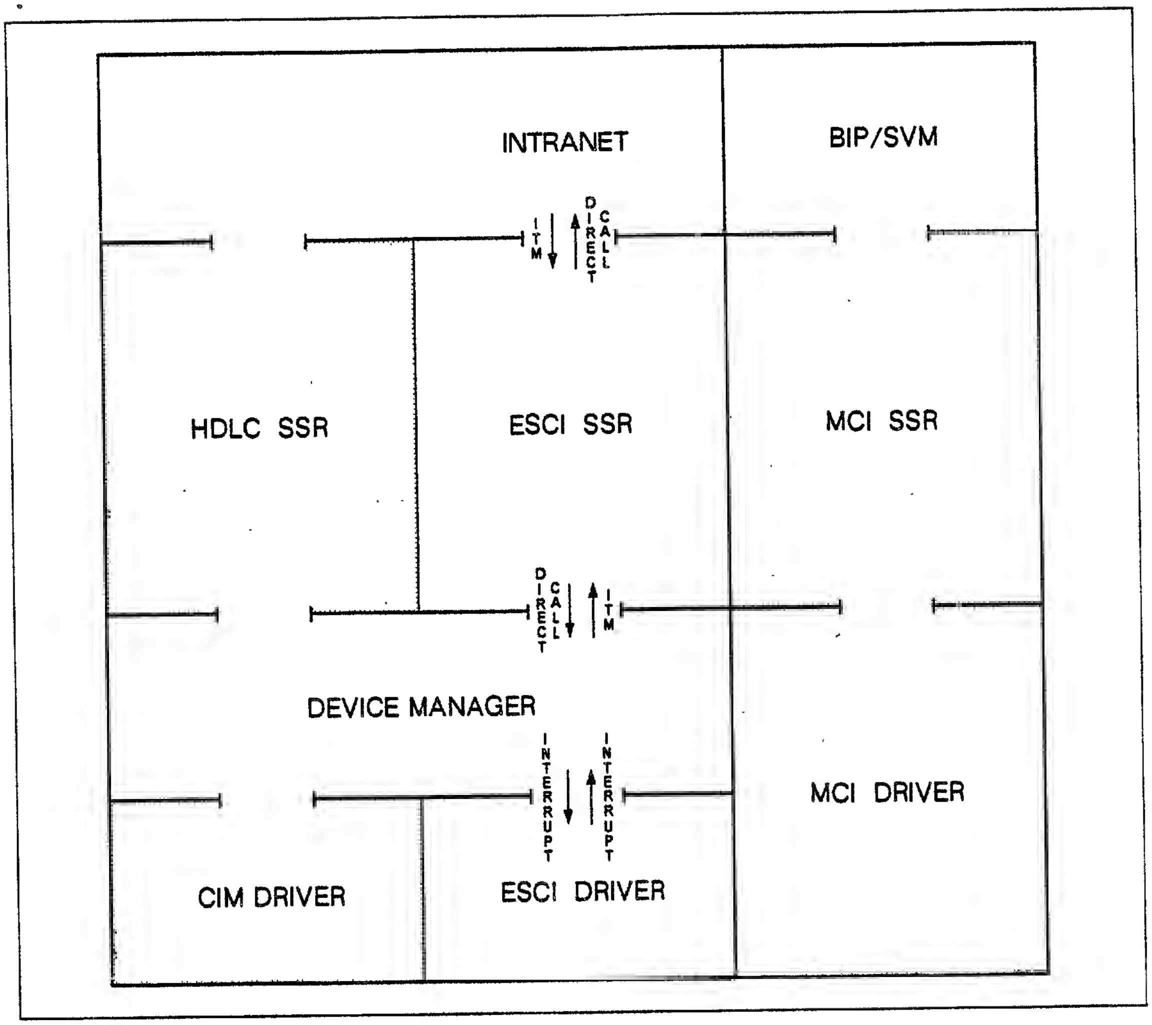


Figure 19-1. ESCI SSR Interfaces

ESCI Initialization and Shutdown

A command processor (DEFINE_ETHER_TRUNK) creates and initializes the link information block (LIB). A command processor (DEFINE_ETHER_NET) creates and initializes the network information block (NIB). The ESCI SSR starts executing as a task. The ESCI SSR executes its initialization procedure and puts itself into an idle state, waiting for a startup ITM.

The command processor (DEFINE_ETHER_NET) sends a startup ITM to the ESCI SSR task. The startup ITM causes the ESCI SSR to initialize the ESCI driver and to change its state. After initialization and startup completes, the ESCI SSR returns an ITM, reporting the success or failure of the startup ITM, to the 3A command processor. The ESCI SSR also returns the status of the link to the Intranet layer whenever link status changes.

A command processor issues a shutdown intertask message to the ESCI SSR. After receiving a shutdown intertask message, the ESCI SSR completes transmission of any frames en route to either the ESCI or the Intranet layer. The ESCI SSR accepts no additional frames from Intranet. Frames already received are transferred to Intranet. Frames being received are discarded. After the ESCI Driver completes shutdown, the ESCI SSR sends an ITM, reporting the link status change, to the requester.

ESCI SSR Data Structure

The ESCI SSR and Intranet share a common data structure, the LIB. This subsection discusses the LIB as used by the ESCI SSR.

Link Information Block

A command processor creates the ESCI LIB during ESCI initialization and startup. The command processor issues a startup intertask message that passes the address of the ESCI LIB to the ESCI SSR. The ESCI LIB contains pointers to the procedures with which ESCI SSR provides services to the Intranet. The ESCI LIB consists of the following sections:

- Physical layer (L1) section that identifies the link to be a specific Ethernet trunk.
- Link layer (L2) section that provides the information needed for the ESCI SSR to control operation of the link.
- Network layer (L3) section that provides the information needed for maintaining the data queue to the Intranet, the Intranet interface procedures, and other configuration parameters.

Constants and Common Types, later in this chapter, contains a CYBIL definition of the LIB record.

Services Required

This section summarizes the services required by the ESCI SSR from the Device Manager (DVM) and the ESCI Driver.

The ESCI SSR requires the services of the DVM and the physical layer software (ESCI Driver) to establish, maintain, and disconnect the physical link between the Ethernet trunk and CDCNET. The ESCI Driver provides the following services to the ESCI SSR through the Device Manager.

- Startup service initializes a particular ESCI.
- Shutdown service removes from service a particular ESCI.
- Transmit service transmits data on ESCI.
- Data reception service receives data on ESCI.
- Statistics reporting service reports physical layer statistics to the ESCI SSR.

ESCI Driver Interface

The DVM provides the services required by the ESCI SSR to communicate with the ESCI Driver.

- The ESCI SSR issues a direct call, requesting a particular service, to the DVM.
- The DVM processes the call, creates a command packet, and places the packet on the command ring to the ESCI.
- The ESCI Driver takes the command packet from the command packet ring, processes the command packet, and returns a status packet, reporting to the DVM the success or lack of success in processing the command.
- The DVM issues a status ITM to the ESCI SSR, communicating the information in the status packet.
- The ESCI SSR processes the status ITM and takes appropriate action based on the current link status.

Because the DVM and ESCI Driver have no interfaces available to the gateway software writer, the DVM and ESCI Driver will not be further described in this manual.

Refer to the Systems Programmer's Reference Manual, Volume 1, Base System Software, for additional Device Manager information.

Services Provided

This section discusses how the ESCI SSR uses the ESCI LIB data structure to provide services to the Intranet layer. Constants and Common Types gives the CYBIL definition of the ESCI LIB data structure. The services provided include the following:

- Data request service
- Indication services

The ESCI SSR provides these services through the LIB. The ESCI SSR uses the LIB to call the Intranet data request procedure and indication procedures. The ESCI SSR passes information to the Intranet layer, enabling data request, data indication, and status indication operations.

Data Request Service

The ESCI SSR provides a data request service to the Intranet layer. The data request service allows the Intranet layer to transfer frames from the Intranet 3A output queue to the ESCI SSR.

Data Request Service

The data request service allows the Intranet layer to transfer frames to the ESCI SSR from the Intranet 3A output queue. The service queues the data and provides the Intranet layer with the information needed to dequeue and transfer the data.

When the Intranet layer detects that the Intranet 3A output queue is empty, the Intranet layer passes a value of NIL to the ESCI SSR and gives the SSR_SLEEPING field of the LIB record a value of TRUE. When the Intranet layer detects an entry in the Intranet output queue, the Intranet layer gives the SSR_SLEEPING field a value of FALSE, and issues an ITM to the ESCI SSR, requesting that the ESCI SSR become active.

Data Request Procedure

The data request procedure allows the ESCI SSR to obtain the next frame to be transmitted over the Ethernet interface. The ESCI SSR calls the SSR_DATA_REQ_PROC to request frames from the Intranet 3A output queue. The ESCI SSR obtains the address of this procedure from the ESCI LIB. The ESCI SSR dequeues the data and passes the data to the ESCI. The data request procedure call is defined below:

```
TYPE
    ssr_data_req_proc_type = ^procedure (
    p: ^cell;
    VAR q: buf_ptr
    ),
```

p: ^cell

The pointer to the LIB for the ESCI interface.

q: buf_ptr

The pointer to the frame that the Intranet layer passes to the ESCI SSR.

Indication Services

The ESCI SSR indication services allow the ESCI SSR to inform the Intranet layer that a frame containing data or status information is available for transfer to Intranet over the ESCI. These indication services include:

- Data indication service
- Status indication service

Data Indication Service

The data indication service allows the transfer of data from the ESCI to the Intranet layer.

SSR_DATA_IND_PROC Procedure

This procedure allows the ESCI SSR to indicate to the Intranet layer that a frame has been received from the ESCI and to transfer the frame to the Intranet layer. The ESCI SSR calls the SSR_DATA_IND_PROC through the ESCI LIB. The data indication procedure call is defined below.

```
TYPE
    ssr_data_ind_proc_type = ^procedure(
    p: ^cell
    VAR q: buf_ptr
    ),
```

p: ^cell

The pointer to the LIB for the ESCI.

q: buf_ptr

The pointer to the frame received from the ESCI.

Status Indication Service

The status indication service allows the Intranet layer to receive status information about the current status of the Ethernet trunk.

SSR_STATUS_IND_PROC Procedure

This procedure allows the ESCI SSR to indicate the status of the link to the Intranet layer. The ESCI SSR calls the SSR_STATUS_IND_PROC procedure through the ESCI LIB. The status indication procedure call is defined below:

```
TYPE
    ssr_status_ind_proc_type = ^procedure (
    p: ^cell;
    s: link_status_type
    );
```

p: ^cell

The pointer to the LIB defining the link between the data link layer and a specific Ethernet trunk.

s: link_status_type

This parameter indicates the current status of the link. The parameter has the following values:

Field Value	Meaning
link_uninitialized	Link has not been initialized and is not available for traffic.
link_initializing	Link has requested initialization data through the bootstrap data unit mechanism.
link_up	Link is available for user traffic.
link_idle	Link is physically available, but traffic is in the process of being suspended.
link_not_available	Link is logically not available, possibly in use by diagnostics.
link_processing_up	Link is in the process of starting up.
link_processing_idle	Link is in the process of being idled.
link_processing_term	Link is in the process of being terminated.
link_down	Link is physically not available.

Constants and Common Types

The ESCI SSR uses the following common types when providing services to its users.

Common Types

Common deck CMCLIBS

```
esci_lib_type = record
    l3: l3_lib_type,
    l2: l2_lib_type,
    l1: l1_lib_type,
recend
```

Common deck CMDLIB1

```
l1_esci_lib_type = record
    configuration_table:      esci_config_tbl_type,
    station_addr:             ethernet_address_type,
    multicast_list:           esci_multicast_list_type
recend
```

Common deck CMCLIB2

```
l2_esci_lib_type
    card_slot:                0 .. 7,
    max_frame_size:           64 .. 1514
recend
```

Common deck CMCLIB3

```
l3_lib_type = record
    nib_ptr:                  ^cell,
    output_qcb:               qcb@,
    nxt_lib_ptr:              ^3b_lib_type,
    link_status:              link_status_type,
    ssr_task_id:              taskid
    ssr_tracing,
    ssr_collecting_stats,
    ssr_sleeping:             boolean,
    ssr_sleeping_lock,
    ssr_data_req_proc:        ssr_data_req_proc_type,
    ssr_data_ind_proc:        ssr_data_ind_proc_type,
    ssr_status_ind_proc:      ssr_status_ind_proc_type,
    link_type:                owner_type,
    trunk_name:               clt$name,
    next_linked_lib:          ^cell,
    lib_defined:              boolean
recend
```

```
link_status_type = (
    link_uninitialized,
    link_initializing,
    link_up,
    link_idle,
    link_not_available,
    link_processing_up,
    link_processing_idle,
    link_processing_term,
    link_down)
```

High-Level Data Link Channel Stream Service Routine 20

High-Level Data Link Channel Stream Service Routine 20

This chapter discusses the high-level data link channel (HDLC) stream service routine (SSR). Topics covered are:

- An overview of the HDLC SSR, including the HDLC SSR user interface and initialization of the HDLC SSR.
- The services required by the HDLC SSR.
- The services provided by the HDLC SSR to its user.
- A list of the CYBIL constants and common types used by the HDLC SSR.

Overview

The HDLC SSR supports interconnection to bit-synchronous, point-to-point communication lines between two device interfaces (DI). The HDLC SSR also provides link-level support for X.25 interconnection to public data networks (PDN). The HDLC SSR executes as a task on the main processor board (MPB) of the DI.

HDLC SSR User Interfaces

The HDLC subsystem, consisting of the HDLC SSR, the Device Manager, and the HDLC Driver, provides the software interface between the Intranet layer and the HDLC trunk. Figure 20-1 shows the interfaces between the HDLC SSR and its user, and the interfaces between the HDLC SSR and the lower layer that provides services to the HDLC SSR (through the Device Manager).

This chapter refers to the data units transferred over the HDLC interface as frames.

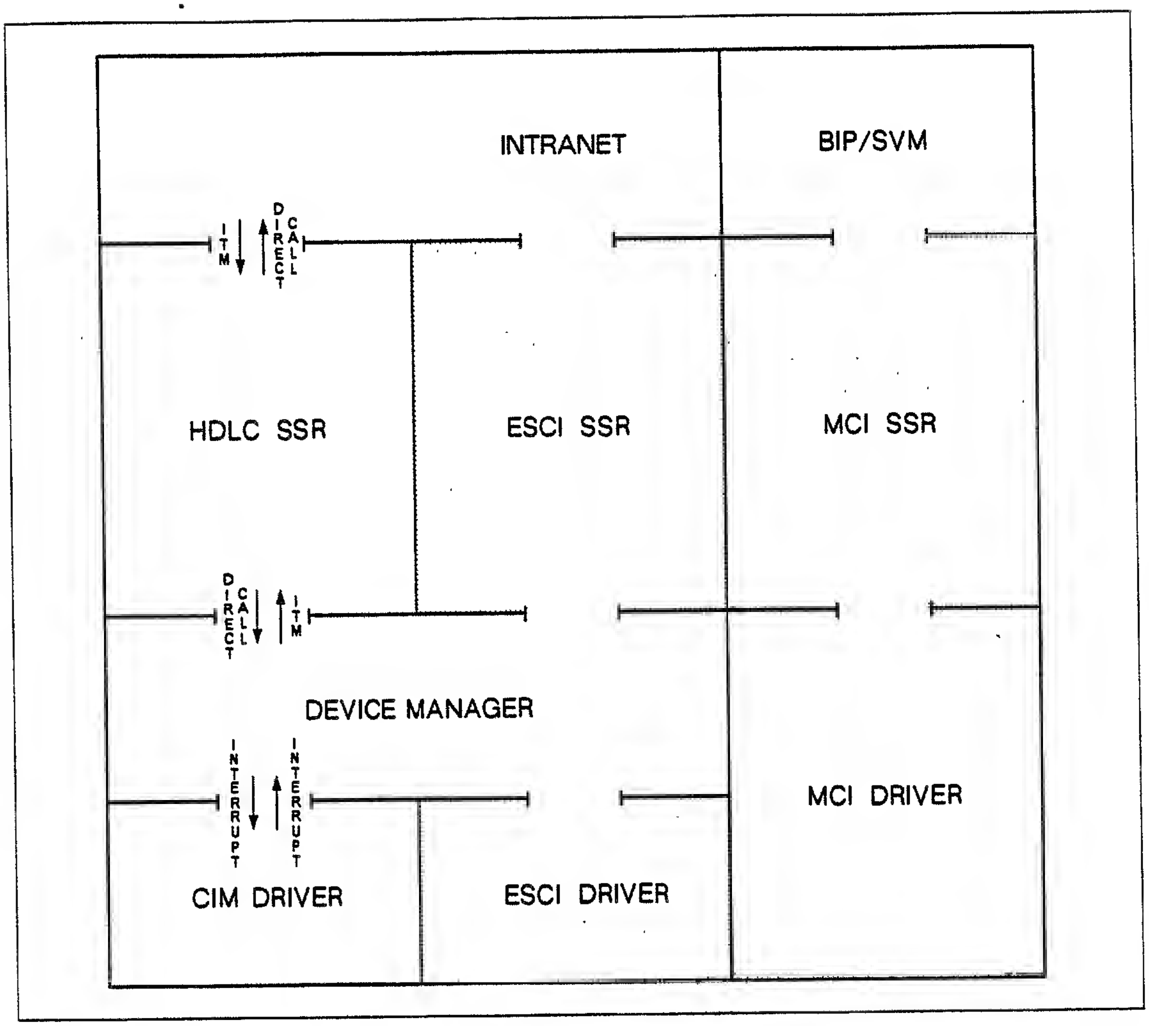


Figure 20-1. HDLC SSR Interfaces

HDLC SSR Initialization

A command processor (DEFINE_HDLC_TRUNK) creates and initializes the link information block (LIB). A command processor creates and initializes the network information block (NIB). The HDLC SSR starts executing as a task. The HDLC SSR executes its initialization procedure and puts itself into an idle state, waiting for a startup ITM.

The command processor (DEFINE_HDLC_NET) also sends a startup ITM to the HDLC SSR task. The startup ITM causes the HDLC SSR to initialize the HDLC driver and to change its state. After initialization and startup completes, the HDLC SSR returns an ITEM reporting the success or failure of the startup ITM. The ESCI SSR also returns the status of the link to the Intranet layer whenever link status changes.

The Device Manager generates an interrupt to the HDLC Driver and returns an intertask message to the HDLC SSR, indicating success or failure of the interrupt.

The HDLC SSR then returns current link status to the Intranet layer.

HDLC SSR Data Structure

The HDLC SSR and the Intranet layer share a common data structure, the link information block (LIB). This section discusses the LIB as used by the HDLC SSR.

Link Information Block

The LIB is created during configuration. During configuration, a command processor configures the HDLC subsystem (HDLC SSR, Device Manager, HDLC trunk), creating a LIB for each trunk. Each LIB contains pointers to the procedures with which the HDLC SSR provides services to the Intranet layer.

- Physical layer (L1) section that identifies the link to a specific HDLC trunk.
- Link layer (L2) section that provides the information needed for the HDLC SSR to control operation of the link.
- Network layer (L3) that provides the information needed for maintaining the data queue to the Intranet layer, the Intranet interface procedures, and other configuration parameters.

The LIB may also provide information needed for a particular subset or superset of a particular type of configuration. For example, the HDLC LIB provides the characteristics of the X.25 extension needed to support connection to an X.25 PDN.

Constants and Common Types contains a CYBIL definition of the LIB record.

Services Required

This subsection summarizes the services required by the HDLC SSR from the Device Manager and the HDLC Driver.

The HDLC SSR requires the services of the Device Manager and the physical layer software (HDLC Driver) to establish, maintain, and disconnect the physical link with the HDLC SSR in a remote system. The HDLC SSR issues direct calls to the Device Manager. The HDLC Driver provides the following services to the HDLC SSR through the Device Manager.

- Startup service initializes a particular HDLC connection.
- Shutdown service removes from service a particular HDLC remote station.
- Heartbeat service detects whether the HDLC remote station is active.
- Transmit service transmits data on the HDLC.
- Data reception service receives data on the HDLC trunk.
- Statistics reporting service reports physical layer statistics to the HDLC SSR.

HDLC Driver Interface

The Device Manager provides the services required by the HDLC SSR to communicate with the HDLC Driver.

- The HDLC SSR issues a direct call, requesting a particular service, to the Device Manager.
- The Device Manager processes the call, creates a command packet, and places the packet on the command ring to the HDLC Driver.
- The HDLC Driver takes the command packet from the command packet ring; processes the command packet; and returns a status packet, reporting to the Device Manager the success or lack of success in processing the command.
- The Device Manager issues a status intertask message to the HDLC SSR, communicating the information in the status packet.
- The HDLC SSR processes the status intertask message and takes the appropriate action based on the current link status.

Because the Device Manager and HDLC Driver have no external interfaces available to a gateway software writer, they will not be described in this manual.

Refer to the Systems Programmer's Reference Manual, Volume 1, Base System Software, for additional Device Manager information.

Services Provided

This section describes the services provided by the HDLC SSR to the Intranet layer. It discusses how the HDLC SSR uses the HDLC LIB data structure to provide these services. Constants and Common Types gives the CYBIL definition of the HDLC LIB data structure. The services provided include the following:

- Data request service
- Indication services

The HDLC SSR provides these services through the LIB. The HDLC SSR uses the LIB to call Intranet data request and indication procedures. The HDLC SSR passes information to the Intranet layer, enabling data request and status indication operations.

Data Request Service

The HDLC SSR provides a data request service to the Intranet layer. The data request service allows the Intranet layer to transfer frames to the HDLC remote station from the Intranet 3A output queue.

Data Request Service

This service allows the Intranet layer to transfer frames to the HDLC remote station. The service queues the data and provides the Intranet layer with the information needed to dequeue and transfer the data. The HDLC SSR provides the data request service by issuing a direct call to the Intranet data request procedure SSR_DATA_REQ_PROC.

Data Request Procedure

This procedure allows the HDLC SSR to obtain the next frame to be transferred from the Intranet 3A output queue. The HDLC SSR calls the SSR_DATA_REQ_PROC to request frames from the Intranet 3A output queue. The HDLC SSR obtains the address of this procedure from the HDLC LIB. The HDLC SSR then dequeues the data and passes the data to the HDLC remote station. The data request procedure call is defined below:

```
TYPE
    ssr_data_req_proc_type = ^ procedure (
    p: ^ cell;
    VAR q: buf_ptr
    ),
```

p: ^cell

The pointer to the LIB for the HDLC trunk.

q: buf_ptr

The pointer to the frame received from the Intranet layer.

Indication Services

The HDLC SSR indication services allow the HDLC SSR to inform the Intranet layer that data and status information is available for transfer to Intranet. The indication services facilitate the transfer of frames and status information. These indication services include the following:

- Data indication service
- Status indication service

Data Indication Service

The data indication service allows the Intranet layer to receive frames from the HDLC remote station.

SSR_DATA_IND_PROC Procedure

This procedure allows the HDLC SSR to indicate frames have been received from the HDLC trunk and to transfer these frames to the Intranet layer. The HDLC SSR calls the SSR_DATA_IND_PROC through the procedure pointer in the HDLC LIB. The data indication procedure call is defined below:

```
TYPE
    ssr_data_ind_proc_type = ^procedure(
    p: ^cell
    VAR q: buf_ptr
    ),
```

p: ^cell

The pointer to the LIB for the HDLC remote station from which the frames are to be transferred.

q: buf_ptr

The pointer to the frame from the HDLC remote station.

Status Indication Procedure

The status indication service allows the Intranet layer to receive updated information about the link between the HDLC remote station and the HDLC SSR. The HDLC SSR provides this service by issuing a direct call to the Intranet status indication procedure SSR_STATUS_IND PROC.

SSR_STATUS_IND_PROC Procedure

This procedure allows the HDLC SSR to indicate the status of the link between the Intranet layer and the physical layer (remote station) of the HDLC subsystem. The address of this procedure is provided to the HDLC SSR through the LIB defining the specific link to the remote station.

```
TYPE
    ssr_status_ind_proc_type = ^procedure (
    p: ^cell;
    s: link_status_type
    );
```

p: ^cell

The pointer to the LIB defining the link between the Intranet layer and a specific remote station.

s: link_status_type

This parameter indicates the current status of the link. The parameter has the following values:

Field Value	Meaning
link_uninitialized	Link has not been initialized and is not available for traffic.
link_initializing	Link has requested initialization data through the bootstrap data unit mechanism.
link_up	Link is available for user traffic.
link_idle	Link is physically available, but traffic is in the process of being suspended.
link_not_available	Link is logically not available, possibly in use by diagnostics.
link_processing_up	Link is in the process of starting up.
link_processing_idle	Link is in the process of being idled.
link_processing_term	Link is in the process of being terminated.
link_down	Link is physically not available.

Constants and Common Types

The HDLC SSR uses the following common types when providing services to its users.

Common Types

Common deck CMCLIBS

```
hdlc_lib_type = record
    l3:                         l3_lib_type,
    l2:                         l2_hdlc_lib_type,
    l1:                         l1_lib_type
recend
```

```
x25_lib_type = record
    l3:                         l3_lib_type,
    l2:                         l2_hdlc_lib_type,
    l1:                         x25_dte_table-type
recend
```

Common deck CMDLIB1

```
l1_lib_type = record
    info:
    command_ring_size:          cim_config_rec_type,
    status_ring_size:           0 .. 255,
    dvm_id:                     integer
recend
```

Common deck CMDLIB2

```
l2_hdlc_lib_type = packed record

    cim_number:                 0 .. 255,
    lim_number:                 0 .. 255,
    port_number:                0 .. 255,
    l_address:                  0 .. 255,
    r_address:                  0 .. 255,
    hdlc_options:               hdlc_option_set
    srej_queue_size:            0 .. 255,
    max_unacked__frames:        0 .. 255,
    min_frame_size:             0 .. 65535,
    max_frames_size:            0 .. 65535,
    i_timer_interval:           0 .. 65535,
    e_timer_interval:           0 .. 65535,
    max_rtx_count:              0 .. 65535,
    rotary_exists:              boolean,
    rotary_group:               clt$name,
    control_block_ptr:          ^cell,
    pdn_id:                     cell
recend
```

Common deck CMDLIB3

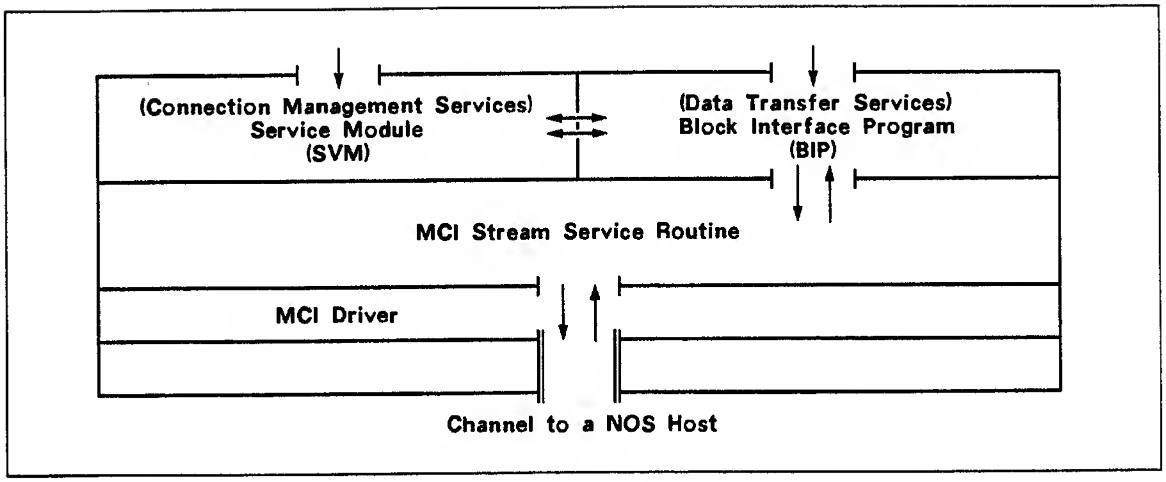
```
l3_lib_type = record
    nib_ptr:                        ^cell,
    output_qcb:                     qcb@,
    nxt_lib_ptr:                    ^3b_lib_type,
    link_status:                    link_status_type,
    ssr_task_id:                    taskid
    ssr_tracing,
    ssr_collecting_stats,
    ssr_sleeping_lock,
    ssr_data_req_proc:              ssr_data_req_proc_type,
    ssr_data_ind_proc:              ssr_data_ind_proc_type,
    ssr_status_ind_proc:            ssr_status_ind_proc_type,
    link_type:                      owner_type,
    trunk_name:                     clt$name,
    next_linked_lib:                ^cell,
    lib_defined:                    boolean
recend

link_status_type = (
    link_uninitialized,
    link_initializing,
    link_up,
    link_idle,
    link_not_available,
    link_processing_up,
    link_processing_idle,
    link_processing_term,
    link_down)
```

Interfaces to Non-CDNA Systems

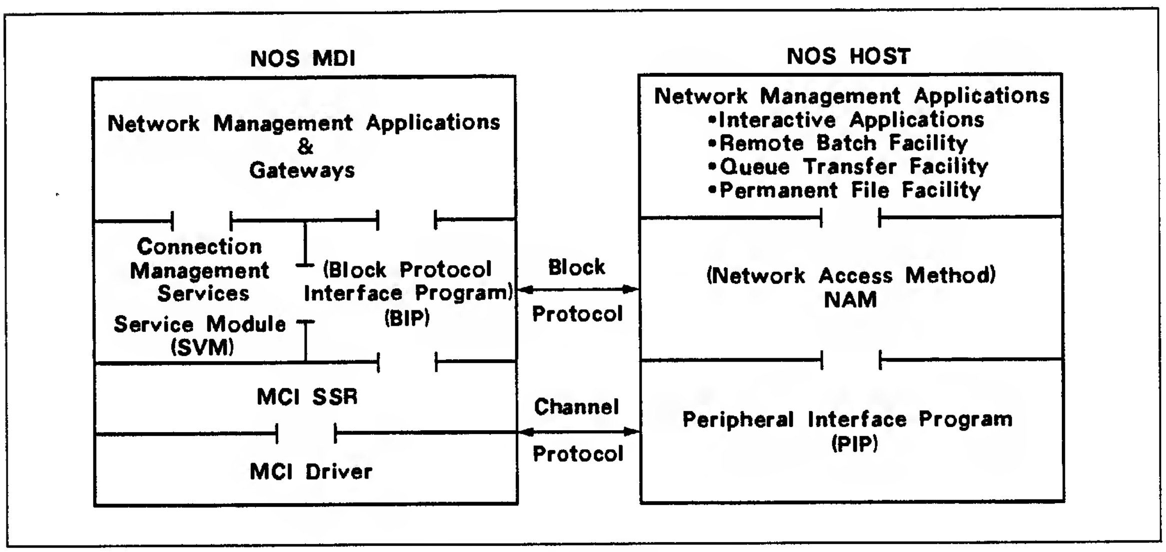
Part IV describes interfaces to non-CDNA systems. CDCNET interface software consists of various software components that enable CDCNET DIs and hosts to communicate with other hosts and networks which do not support CDNA. This part of the manual describes the following two interfaces:

- Interface to NOS hosts
- Interface to X.25 networks

(Continued)



NOS Host Interface Software



CDCNET MDI/NOS Host Relationship

Chapters 21 and 22 describe BIP and SVM, respectively. The MCI SSR is described in chapter 18. The MCI Driver is not described in this manual because it has no user interfaces.

Interface to NOS Hosts

The NOS host interface is provided to enable CDCNET systems to communicate with applications on the NOS hosts.

Traditionally, NOS hosts have interfaced with the Communication Control Program (CCP) in a front-end processor (255x). Four interface programs, collectively referred to as the Network Access Method (NAM), provide the means for NOS applications to access the communication network.

CDCNET provides a NOS host interface so that the existing NAM applications can be integrated with CDCNET. In addition, CDCNET uses the NOS host interface to communicate with NAM applications that have been specially developed to provide CDCNET network management capabilities.

The CDCNET NOS interface is provided by preserving as much of the NOS/CCP protocols as possible. It is responsible for mapping the NOS Network Products connection and Block Protocol with the CDNA Connection and Data Transfer Protocol. This interface resides in a Mainframe Device Interface (MDI) connected to a NOS host and consists of the following three software components:

- The Block Protocol Interface Program (BIP)
- The Service Module (SVM)
- The Mainframe Channel Interface (MCI) Stream Service Routine (SSR) and MCI Driver

BIP and SVM use the Network Block Protocol to communicate to the NAM software in the NOS host. Data between NAM and the NOS host interface is transferred over a logical connection. Logical connections are dynamically initiated and terminated. These connections are established on logical links and each connection is given a unique connection number. An MDI connected to a NOS host is capable of supporting multiple logical links, but each logical link requires a separate CYBER channel and a separate MCI board in the MDI.

BIP depends upon the services of the MCI SSR and the MCI Driver for the physical transmission of data across the coupler to the channel connecting the NOS host. This physical interface is new, and requires a new channel protocol which is supported on the host by new overlays in the NAM peripheral interface program (PIP).

CDCNET provides Network Products gateways which use the NOS host interface to connect CDCNET interactive terminals, batch terminals, and applications to the NAM applications on the NOS host.

CDCNET also provides three MDI applications which use the NOS host interface to communicate with peer NAM applications. The peer applications provide CDCNET network management functions like file access, network logging, and network operation.

The following figures show the NOS host interface software, and the relationship between a CDCNET MDI and a NOS host, respectively.

(Continued on other side)

Block Protocol Interface Program (BIP) 21

Block Protocol Interface Program (BIP) 21

This chapter discusses:

- An overview of the Block Protocol Interface Program (BIP).
- The services required by BIP.
- The services provided by BIP.
- Constants and common types used in BIP's service requests.

Overview

BIP is a software component that helps CDCNET applications and gateways connect to the Network Access Method (NAM) in a NOS (Network Operating System) host. BIP is part of a special interface that is provided to connect a NOS host to CDCNET, because NOS host systems do not support CDCNET architecture. BIP resides only in a mainframe device interface (MDI) or a mainframe terminal interface (MTI) that is connected to the host computer system running NOS.

The interface, of which BIP is a part, is designed to emulate the Communication Control Program (CCP) in the 255x, and consists of the following three components:

- BIP
- Service Module (SVM)
- Mainframe Channel Interface (MCI)

For more information on NAM, the Network Products and the CCP, see the NOS 2 Network Definition Language Reference Manual.

The primary responsibility of BIP is to provide an efficient data transport service between local MDI applications and NOS applications. It also supports a local interface with SVM to exchange connection management messages between SVM and NAM.

BIP depends on the services of the Mainframe Channel Interface (MCI) Stream Service Routines (SSR) for the actual physical transmission of network blocks across the coupler to the channel connected to the NOS host. BIP and SVM add to the services of the MCI SSR to provide a transport-like service to local MDI applications and gateways. While SVM provides services to establish and terminate Network Products connections, BIP provides services for data transfer across an established Network Products connection. Figure 21-1 illustrates the interrelationship of these three software products.

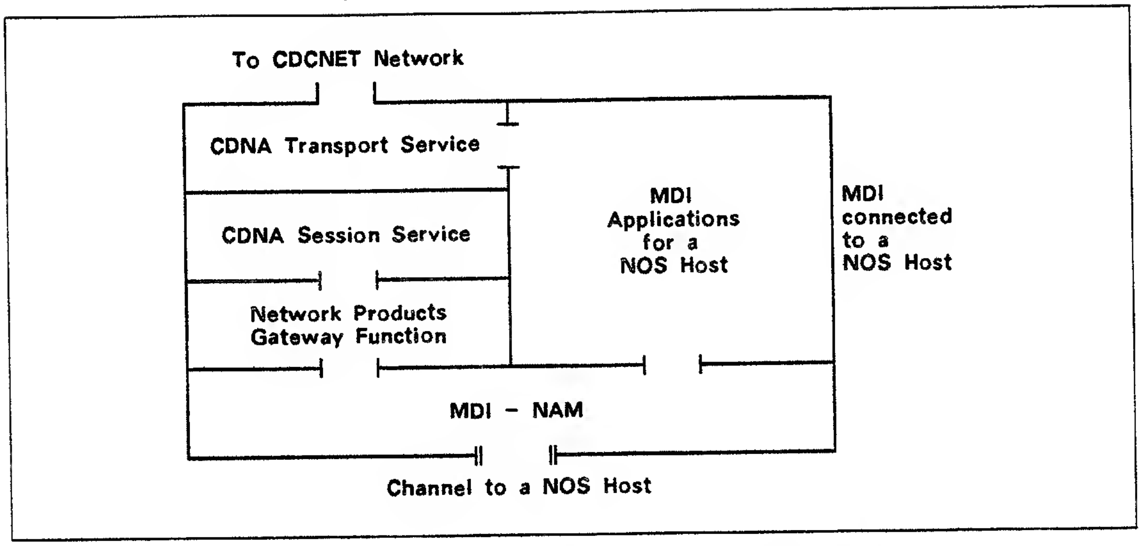


Figure 21-1. NOS Host Interface Software

BIP's main responsibility is to transport data from a user on CDCNET to a destination in the NOS host. The data is transmitted in the form of blocks. All service requests from the users come to BIP through a direct call interface through the externally referenced (XREF) procedure, APPL_REQ. BIP sends indications or responses to these requests through a procedure whose address is passed through the APPL_IF parameter, when the user establishes a connection with the SVM. Details on connections are given in the Service Module (SVM) chapter in this manual.

Details on BIP services and the user's procedure whose address is passed in the APPL_IF parameter, are described next.

Services Required

This section briefly describes the services of SVM and MCI, with which BIP works to help CDCNET connect to a NOS host. Details on each software component are covered in other chapters of this manual.

Service Module (SVM)

SVM and BIP together use the Network Products' Block Protocol to communicate with NAM in the NOS host. SVM is responsible for establishing the connections on which data is transferred. See chapter 22 in this manual for more information.

Mainframe Channel Interface (MCI)

BIP depends on the services of the MCI SSR for the actual physical transmisson of the Network Blocks across the coupler to the channel connected to the NOS host. See chapter 18 in this manual for more information.

Services Provided

This section defines the external services provided by BIP to its users. All service requests to BIP are made through the APPL_REQ procedure. APPL_REQ is an externally referenced (XREF) procedure. The indications to these service requests are sent to a procedure, whose address is sent through the APPL_IF parameter when the connection was first established with the SVM. BIP users include the common deck C7DAPRQ in their calling modules. Common deck C7DAPRQ contains this procedure declaration:

```
PROCEDURE [XREF] appl_req (svm_cepid: ^cell;
      service_req: bip_request;
  VAR data_ptr: buffer;
  VAR status: boolean );
```

The common deck C7DAPRQ contains calls to other common decks that contain BIP service requests and type definitions.

The user specifies a value in the SERVICE_REQ parameter in the above procedure which determines the particular BIP service being requested.

In the following pages, the APPL_REQ procedure is defined first, followed by the description of various service requests that are specified through this procedure. These are followed by the description of the user's procedure and the various service indications that BIP sends back to the user. Constants and common types are given at the end of the chapter.

BIP's services can be functionally divided into two groups.

- Service Requests
- Service Indications

Service Requests

The following are the various types of service requests made to BIP. The user specifies one of these requests in the APPL_REQ procedure.

APPL_REQ

This procedure is called by the BIP users to send service requests to BIP after a connection is established. APPL_REQ is responsible for validating the service request and then generating the appropriate Network Block.

Comdeck **C7DAPRQ**

Format **APPL_REQ (svm_cepid, service_req, data_ptr, status)**

Input

svm_cepid: ^cell

The CEPID shared by SVM and BIP. This identifier uniquely describes the connection to BIP. Users receive the address of this identifier when they establish a connection with SVM. See SV_CALL_CONFIRM indication in chapter 22.

service_req: bip_request

The key parameter which must contain one of the following values:

BLK_REQ

QBLK_REQ

MSG_REQ

QMSG_REQ

CMD_REQ

ICMD_REQ

ICMDR_REQ

BRK_REQ

RST_REQ

SFD_REQ

CFD_REQ

CANCEL_INPUT

data_ptr: buffer

This parameter contains 1 through 2043 bytes of optional data that can be sent through the BLK_REQ, MSG_REQ, CMD_REQ, ICMD_REQ, and BRK_REQ requests.

Output

status: boolean

This parameter indicates whether or not the user's service request was processed.

TRUE	The service request was processed successfully.
FALSE	The service request was not processed due to error(s). The user is responsible for processing its error(s) and releasing the buffer chain containing the data. The error messages are sent to a log file.

The following pages describe the various service requests that are specified through the APPL_REQ procedure. In these descriptions, the parameters described in the APPL_REQ procedure are listed again, but the explanation to each of these parameters is not repeated unless there is some unique information for that particular parameter.

BLK_REQ

This request allows the BIP user to send the first or intermediate block of a multiblock message over a previously established Network Products connection.

Input The user sets up the following fields as input in the APPL_REQ procedure:

svm_cepid: ^cell

service_req: bip_request

This parameter must contain the following value:

BLK_REQ

data_ptr: buffer

This parameter contains 1 through 2043 bytes of data to be sent to a peer user in the NOS host.

Output **status: boolean**

QBLK_REQ

This request allows the user to send qualified data on an application-to-application connection over a previously established Network Products connection.

Input The user sets up the following fields as input in the APPL_REQ procedure:

svm_cepid: ^cell

service_req: bip_request

This parameter must contain the following value:

QBLK_REQ

data_ptr: buffer

This parameter contains 1 through 2043 bytes of intermediate qualified message data to be sent to a peer user in the NOS host.

Output **status: boolean**

MSG_REQ

This request allows the BIP user to send either the last block of a multiblock data message or the only block of a data message to be sent or received over a previously established Network Products connection.

Input The user sets up the following fields as input in the APPL_REQ procedure:

svm_cepid: ^cell

service_req: bip_request

This parameter must contain the following value:

MSG_REQ

data_ptr: buffer

This parameter contains 1 through 2043 bytes of data to be sent to a peer user in the NOS host.

Output **status: boolean**

QMSG_REQ

This request allows the user to send either the last block of a multiblock, qualified data message or the only block of a qualified data message to be sent or received over a previously established Network Products connection.

Input The user sets up the following fields as input in the APPL_REQ procedure:

svm_cepid: ^cell

service_req: bip_request

This parameter must contain the following value:

QMSG_REQ

data_ptr: buffer

This parameter contains 1 through 2043 bytes of data to be sent to a peer user in the NOS host.

Output **status: boolean**

CMD_REQ

This request allows the user to communicate with a peer user in the NOS host through a communication path. This path is outside the data stream but is synchronous with the data stream block delivery. The Command (CMD) block can be used either as a marker in the stream to indicate the point at which an action took place, or can be used to send an instruction to its receiver.

Input The user sets up the following fields as input in the APPL_REQ procedure:

svm_cepid: ^cell

service_req: bip_request

This parameter must contain the following value:

CMD_REQ

data_ptr: buffer

This parameter contains 1 through 2043 bytes of data to be sent to a peer user in the NOS host.

Output **status: boolean**

ICMD_REQ

This request allows the BIP user to communicate with a peer user in the NOS host through a communication path. This path is outside the data stream and bypasses normal message delivery sequence. This service is used by BIP users to send one byte of expedited data to a peer across an established Network Products connection.

Input The user sets up the following fields as input in the APPL_REQ procedure:

svm_cepid: ^cell

service_req: bip_request

This parameter must contain the following value:

ICMD_REQ

data_ptr: buffer

This parameter contains one byte of expedited data to be sent to the peer user.

Output **status: boolean**

NOTE

Only one expedited data message can be outstanding at a time.

ICMDR_REQ

This request allows the BIP user to send an acknowledgement to expedited data that was previously received from a peer user in the NOS host, across an established Network Products connection.

Input The user sets up the following fields as input in the APPL_REQ procedure:

svm_cepid: ^cell

service_req: bip_request

This parameter must contain the following value:

ICMDR_REQ

data_ptr: buffer

This parameter is not used in this request.

Output **status: boolean**

BRK_REQ

This request allows the BIP user to process interruptions on the data stream. This procedure is used to indicate that the user is temporarily unable to receive BLK, MSG, or CMD indications on an establised Network Products connection. BIP discards all subsequent indications from the peer until an RST_IND is received from the BIP user. This service is used for A-A connections only.

Input The user sets up the following fields as input in the APPL_REQ procedure:

svm_cepid: ^cell

service_req: bip_request

This parameter must contain the following value:

BRK_REQ

data_ptr: buffer

This parameter can be used to send 0 through 255 bytes of user data.

Output **status: boolean**

RST_REQ

This request allows the BIP user to identify the point in the data stream at which data blocks are valid after a BRK_IND was issued. This procedure is used to indicate that the BLK_REQ, MSG_REQ, and CMD_REQ requests following an RST_REQ are valid and should be transmitted to a peer in the NOS host. An RST_REQ arrives after a BRQ_IND is received by the peer in the NOS host.

Input The user sets up the following fields as input in the APPL_REQ procedure:

svm_cepid: ^cell

service_req: bip_request

This parameter must contain the following value:

RST_REQ

data_ptr: buffer

This parameter is not used in this request.

Output **status: boolean**

SFD_REQ

This request allows the user to inform NAM (through BIP) to hold back block, message, and command types of indications. Although this request will eventually be honored, the user must be able to accept a few block, message, and command indications (equal to the Downline Block Limit set by NAM) that will arrive for it after a suppress forward data request is made.

Input The user sets up the following fields as input in the APPL_REQ procedure:

svm_cepid: ^cell

service_req: bip_request

This parameter must contain the following value:

SFD_REQ

data_ptr: buffer

This parameter is not used in this request.

Output **status: boolean**

CFD_REQ

This request allows the BIP user to stop flow control that it had previously requested. When BIP receives this request, it checks the buffer state and sends all the requests that were held back when the flow control request was made.

Input The user sets up the following fields as input in the APPL_REQ procedure:

svm_cepid: ^cell

service_req: bip_request

This parameter must contain the following value:

CFD_REQ

data_ptr: buffer

This parameter is not used in this request.

Output **status: boolean**

CANCEL_INPUT

This request allows the BIP user to request that any data within BIP's input queue, for a particular connection, be discarded. This service has been provided so that end-user breaks are properly handled by the IVT gateway.

Input The user sets up the following fields as input in the APPL_REQ procedure:

svm_cepid: ^cell

service_req: bip_request

This parameter must contain the following value:

CANCEL_INPUT

data_ptr: buffer

This parameter is not used in this request.

Output **status: boolean**

Service Indications

The following are the various indications to service requests made by BIP users. These indications are sent by BIP to its users, through a procedure whose address is passed through the APPL_IF parameter.

APPL_IF

This parameter contains the address of the user procedure through which indications to service requests are sent by BIP to its users. The address of this procedure is passed to SVM by the user when a Network Products connection is established. See chapter 22 in this manual for details on SVM and the establishment of connections.

Comdeck C7DSVWK

Format **APPL_IF = ^procedure (ucepid, indication, ptr)**

Input

ucepid: ^cell

The CEPID that uniquely describes the connection to the BIP user.

indication: bip_indication

The key parameter which must contain one of the following values:

XFER_IND

BLK_IND

QBLK_IND

MSG_IND

FILLER_IND

QMSG_IND

CMD_IND

ICMD_IND

ICMDR_IND

BRK_IND

RST_IND

SFD_IND

CFD_IND

ptr: buf_ptr

This parameter contains 1 through 2043 characters of optional data that can be sent with BLK_IND, MSG_IND, CMD_IND, ICMD_IND, and BRK_IND indications.

XFER_IND

This indication informs the BIP user that the Network Block Protocol has been initiated and that the connection to the peer user in the NOS host has been established and can now support data transfer. This indication must be received by the BIP user before any other BIP services can be requested.

Input BIP sets up the following fields as input in the user procedure:

ucepid: ^cell

indication: bip_indication

This parameter must contain the following value:

XFER_IND

data_ptr: buffer

This parameter is not used in this request.

BLK_IND

This indication informs the BIP user that the first or intermediate block of a multiblock message has arrived on a previously established Network Products connection.

Input BIP sets up the following fields as input in the user procedure:

ucepid: ^cell

indication: bip_indication

This parameter must contain the following value:

BLK_IND

ptr: buf_ptr

This parameter contains 1 through 2043 bytes of data to be sent to the BIP user.

QBLK_IND

This indication allows BIP to inform its user that the first or intermediate block of a qualified multiblock message has arrived on a previously established Network Products connection.

Input BIP sets up the following fields as input in the user procedure:

ucepid: ^cell

indication: bip_indication

This parameter must contain the following value:

QBLK_IND

data_ptr: buffer

This parameter contains 1 through 2043 bytes of data to be sent to the BIP user.

MSG_IND

This indication allows BIP to inform its user that either the last block of a multiblock data message or the only block of a data message has arrived on a previously established Network Products connection.

Input BIP sets up the following fields as input in the user procedure:

ucepid: ^cell

indication: bip_indication

This parameter must contain the following value:

MSG_IND

data_ptr: buffer

This parameter contains 1 through 2043 bytes of data to be sent to the BIP user.

QMSG_IND

This indication allows BIP to inform the user that either the last block of a multiblock, qualified data message or the only block of a qualified data message has arrived on a previously established Network Products connection.

Input BIP sets up the following fields as input in the user procedure:

ucepid: ^cell

indication: bip_indication

This parameter must contain the following value:

QMSG_IND

data_ptr: buffer

This parameter contains 1 through 2043 bytes of data to be sent to the BIP user.

CMD_IND

This indication allows BIP to inform its user that a message outside the data stream has arrived on a previously established Network Products connection.

Input BIP sets up the following fields as input in the user procedure:

ucepid: ^cell

indication: bip_indication

This parameter must contain the following value:

CMD_IND

data_ptr: buffer

This parameter contains 1 through 2043 bytes of data for the BIP user.

ICMD_IND

This indication allows BIP to inform its user that one byte of expedited has arrived for it from a peer across a previously established Network Products connection.

Input BIP sets up the following fields as input in the BIP_IF procedure:

ucepid: ^cell

indication: bip_indication

This parameter must contain the following value:

ICMD_IND

data_ptr: buffer

This parameter contains one byte of expedited data.

ICMDR_IND

This indication allows BIP to inform its user that an acknowledgment to expedited data has arrived on a previously established Network Products connection.

Input BIP sets up the following fields as input in the user procedure:

ucepid: ^cell

indication: bip_indication

This parameter must contain the following value:

ICMDR_IND

data_ptr: buffer

This parameter is not used in this request.

BRK_IND

This indication allows BIP to inform its users that the peer user in the NOS host is unable to receive BLK_REQ, MSG_REQ, or CMD_REQ requests. BIP discards all subsequent requests until an RST_REQ request is received from the user.

Input BIP sets up the following fields as input in the user procedure:

ucepid: ^cell

indication: bip_indication

This parameter must contain the following value:

BRK_IND

data_ptr: buffer

This parameter contains 0 through 255 bytes of information for the user.

RST_IND

This indication allows BIP to inform its user that all BLK_REQ, MSG_REQ, and CMD_REQ requests following an RST_IND are valid transmissions from the peer in the NOS host. An RST_IND arrives after a BRQ_REQ is received by the user.

Input BIP sets up the following fields as input in the user procedure:

ucepid: ^cell

indication: bip_indication

This parameter must contain the following value:

RST_IND

data_ptr: buffer

This parameter contains 0 through 255 bytes of information for the user.

SFD_IND

This indication allows BIP to inform its users to hold block, message, and command requests. This indication is sent to the user in the DI either when the user's upline block limit reaches a threshold level or when BIP terminates a connection because of a NAM protocol error.

Input BIP sets up the following fields as input in the user procedure:

ucepid: ^cell

indication: bip_indication

This parameter must contain the following value:

SFD_IND

data_ptr: buffer

This parameter is not used in this request.

CFD_IND

This indication allows BIP to inform its user to resume the transmission of BLK_REQ, MSG_REQ, and CMD_REQ requests.

Input BIP sets up the following fields as input in the user procedure:

ucepid: ^cell

indication: bip_indication

This parameter must contain the following value:

CFD_IND

data_ptr: buffer

This parameter is not used in this request.

Constants and Common Types

This section lists the common types used by all the procedures described in this chapter. The following types are defined in the common deck C7DBIAR.

Common Types

```
 bip_indication =
( xfer_ind,
  blk_ind,
  msg_ind,
  cmd_ind,
  brk_ind,
  sfd_ind,
  cfd_ind,
  rst_ind,
  icmd_ind,
  icmdr_ind,
  qblk_ind,
  qmsg_ind,)

   bip_proc = ^procedure (ucepid: ^cell;
     indication: bip_indication;
     VAR ptr: buf_ptr)

 bip_request = (
   blk_req,
   msg_req,
   cmd_req,
   brk_req,
   sfd_req,
   cfd_req,
   rst_req,
   icmd_req,
   icmdr_req,
   qblk_req,
   qmsg_req,
   cancel_input)
```

Service Module (SVM) 22

Service Module (SVM) 22

This chapter discusses:

- An overview of the Service Module (SVM).
- The services required by SVM.
- The services provided by SVM.
- Constants and common types used in SVM's service requests.

Overview

SVM is a software component that helps CDCNET applications and gateways connect to the Network Access Method (NAM) in a host computer running NOS (Network Operating System). SVM resides only in a Mainframe Device Interface (MDI) or a Mainframe Terminal Interface (MTI) that is connected to a host computer system running NOS. SVM is part of a special interface that is provided to connect a NOS host to CDCNET, as NOS host systems do not support CDCNET architecture. This interface is designed to emulate the Communication Control Program (CCP) in 255x and consists of the following three software components:

- SVM
- Mainframe Channel Interface (MCI)
- Block Protocol Interface Program (BIP)

MCI and BIP interfaces are described in chapters 18 and 21, respectively, in this manual. For more information on NAM, the Network Products, and the CCP, see the NOS 2 Network Definition Language Reference Manual.

Connections and Connection Establishment

The primary responsibility of the SVM is to establish a logical connection between the applications that reside in the host running NOS and the applications and gateways that reside in the MDI/MTI. Logical connections are established on logical links which are established on the physical channel between the MDI/MTI and NAM in the NOS host.

To establish these connections, SVM uses the Network Block Protocol to communicate with NAM. When the link is up, NAM informs SVM through BIP. SVM, in turn, informs its users that the link is up and that requests for connections can be processed. Likewise, when the link goes down, SVM informs all the users who have a SAP open for that connection that the link is down.

Connections are of two types:

- A-A connections, which are connections between two host applications.
- T-A connections, which are connections between terminals and host applications.

Some requests can be received on either type of connection while others are restricted to one type of connection. See SVM Services, later in this chapter, for more information on the various types of requests that can be sent on these connections.

SVM receives all requests to establish or terminate connections through an intertask message (ITM) handler. Once the ITM handler is initiated, it loops continuously, processing all the ITMs. When users need SVM's services, they first request a Directory translation of SVM's title and obtain its task identifier. The task identifier and service request workcode are sent to the ITM handler. When the ITM handler receives the task identifier and workcode, it calls an appropriate procedure to process them. A value specified in the workcode determines which procedure is called. Details on the various service request workcodes are given in Services Provided later in this chapter.

SVM Services

SVM is used by applications and gateways in MDIs and MTIs to connect to applications residing in NOS hosts. SVM is also responsible for giving the status of the link to its users. SVM's services can be functionally divided into two groups.

- Service Requests
- Service Indications

Details on requests and indications are given in Services Provided later in this chapter.

Services Required

SVM works with BIP to help CDCNET connect to a NOS host. This section briefly describes the services of these software components. Details on each software component are covered in other chapters in this manual.

BIP

SVM, along with BIP, uses the Network Products Block Protocol to communicate with NAM in the NOS host. SVM uses BIP's services to send service messages to NAM. BIP adds the priority bit, the block type, and the connection number to the message before passing it to the MCI SSR. All service messages are sent and received on connection number zero. BIP sends all incoming service messages to SVM through ITMs. See chapter 21 in this manual for more information on BIP.

Directory ME

When SVM is initialized, its title is registered with the Directory ME. SVM users translate this title to get the address of SVM's task identifier. The task identifier is used as an address by the Executive to send all intertask messages from the users to SVM. For more information on the Executive, see the CDCNET Systems Programmer's Reference Manual, Volume 1. Chapter 5 in this manual gives details on Directory ME.

Services Provided

This section defines the external services provided by SVM to its users. As mentioned earlier, all service requests to SVM are intertask messages. Before users can make any service requests, they use the Directory ME's services to translate the SVM title and get its task identifier. The following example shows how the SVM title is translated by the user to obtain the SVM task identifier. For more information on Directory Translation procedures, see chapter 5 in this manual.

```
dir_tcb_init (dir_tcb);
dir_tcb.title_ptr := title_svm;          {title that SVM registers
dir_tcb.search_domain := local_system;
dir_translate_and_wait (dir_tcb, dir_ttcb, status);

IF status=dir_title_found THEN
    svm_task:=dir_ttcb.address.record_ptr;
IFEND;
```

Users include the common deck C7DSVWK in their calling modules. This common deck contains the SVM_ITM record. A value specified in the workcode parameter of this record determines the service being requested. Each workcode value selects a variant of the SVM_ITM record. The following example shows how a gateway program sends a service request to open an SVM SAP.

```
itm.open_wk.who_are_you := i_am_a_gw;        {svm_secondary_workcode type
itm.open_wk.reply_to := i_am_a_gw_task;      {task_id for the component to
                                             {which svm should send
                                             {an open indication
itm.open_wk.svm_if := i_am_a_gw_task;        {task_id for the component to
                                             {which svm should send other
                                             {indications.
itm.open_wk.bip_if := ^bip_service_ind;      {address
```

Following is a CYBIL description of the SVM_ITM record. The constants and common types used in the ITMs are described at the end of this chapter.

```
TYPE
  svm_itm = record
    case workcode: svm_reg .. sv_terminal_accounting_stats of

    = sv_reg =
      reg_wk: record
        reglevel: regulation_levels,
      recend,

    = sv_clear =
      clear_wk: record
        svm_cepid: ^cell,
      recend,

    = sv_call_confirm =
      confirm_wk: record
        svm_cepid: ^cell,
        ucepid: ^cell,
      recend,
```

```
= sv_call =
  call_wk: record
    ucepid: ^cell,
    task: task_ptr,
    cc_if: ^procedure (indication: sv_indication),
    bip_if: ^procedure (ucepid: ^cell;
      indication: bip_indication;
      VAR ptr: buffer),
    text: buf_ptr,
  recend,

= sv_rej =
  rej_wk: record
    svm_cepid: ^cell,
    text: 0 .. 0ff(16),
  recend,

= sv_open =
  open_wk: record
    who_are_you: svm_secondary_workcode,
    reply_to: task_ptr,
    svm_if: task_ptr,
    bip_if: ^procedure (ucepid: ^cell;
      indication: bip_indication;
      VAR ptr: buffer),
  recend,

= sv_close =
  close_wk: record
    who_are_you: svm_secondary_workcode,
    reply_to: task_ptr,
  recend,

= sv_init =
  lib_address: ^mci_lib_type,

{     =sv_cancel=

= sv_down =
  buffer_address: buf_ptr,

= sv_term =
  term_wk: record
    cn: node_range,
  recend,

= sv_call_t =
  call_t_wk: record
    ucepid: ^cell,
    task: task_ptr,
    cc_if: ^procedure (indication: sv_indication),
    bip_if: ^procedure (ucepid: ^cell;
      indication: bip_indication;
      VAR ptr: buffer),
    upline_blk_limit: 0 ... 255,
```

```
      text: buf_ptr,
    recend,

  = sv_clear_t =
    clear_t_wk: record
      svm_cepid: ^cell,
      text: buf_ptr,
    recend,

{ = sv_shutdown =

  = sv_connection_down =
    connection_down_wk: record
      cepid: ^cell,
    recend,

  = sv_terminal_characteristics =
    tchar_wk: record
      svm_cepid: ^cell,
      text: buf_ptr,
    recend,

  = sv_application_accounting_stats =
    aacct_wk: record
      text: buf_ptr,
    recend,

  = sv_terminal_accounting_stats =
    tacct_wk: record
      text: buf_ptr,
    recend,

  casend,
recend;
```

The following pages describe the various types of service requests made to SVM. A value specified in the workcode parameter of the SVM_ITM record determines the service being requested. Each workcode value selects a variant of the SVM_ITM record.

NOTE

The service requests SV_REG, SV_CONNECTION_DOWN, SV_SHUTDOWN, SV_TERM, SV_CANCEL, SV_INIT, and SV_DOWN are part of the SVM_ITM record but unlike other requests, are not exchanged between the user and SVM. Therefore, they are not described in the following pages.

SV_OPEN

This request allows the user to open an SVM SAP. This is the first request the user makes to SVM. Once the SAP is opened, other service requests can be made.

Format

```
open_wk: record
    who_are_you: svm_secondary_code
    reply_to: task_ptr
    svm_if: task_ptr
    bip_if: ^procedure
```

Input

The user sets up the following fields as input in the OPEN_WK record, which is in the SVM_ITM record:

who_are_you: svm_secondary_code

This ordinal identifies the user. See Constants and Common Types later in this chapter for more information.

reply_to: task_ptr

This task identifier is used by SVM to send an indication to the open SAP request.

svm_if: task_ptr

This task identifier is used by SVM to send all indications (except an indication to an open SAP request) to the user.

bip_if: ^procedure

The address of a procedure which is used by BIP to send all indications and responses to its users. See chapter 21 in this manual.

SV_CLOSE

This request allows the user to request that SVM close the SAP.

Format

close_wk: record
who_are_you: svm_secondary_workcode
reply_to: task_ptr

Input

The user sets up the following fields as input in the CLOSE_WK record, which is in the SVM_ITM record:

who_are_you: svm_secondary_workcode

This ordinal identifies the user. See Constants and Common Types later in this chapter for more information.

reply_to: task_ptr

This is the address of the task to which SVM sends an indication to this request. If it is a NIL value, no ITM is sent.

SV_CALL

This request allows the user to request that an application-to-application connection be established with its peer.

Format

call_wk: record
  ucepid: ^cell
  task: task_ptr
  cc_if: ^procedure
  bip_if: ^procedure
  text: buf_ptr

Input

The user sets up the following fields as input in the CALL_WK record, which is in the SVM_ITM record:

ucepid: ^cell

The address of the user's CEPID.

task: task_ptr

This task identifier is used by SVM to send indications.

cc_if: ^procedure

The address of a procedure used by SVM to inform the user that its request to establish a connection was successful. This indication is sent to the user through a direct call procedure rather than through an ITM. This prevents data from reaching the user before the user has been notified that its connection request has been accepted.

bip_if: ^procedure

The address of a procedure which is used by BIP to send all indications and responses to its users. See chapter 21 in this manual for more information on this procedure.

text: buf_ptr

The pointer to the text containing 0 through 255 bytes of protocol-related information. SVM appends this text to the header which contains information from the user.

SV_CALL_CONFIRM

This request allows the user to accept a connection that is initiated by its peer. This request is used for A-A connections only. A CALL_INDICATION from a peer initiates this request, which is used to transfer data from one mainframe to another.

Format

confirm_wk: record
svm_cepid: ^cell
ucepid: ^cell

Input

The user sets up the following fields as input in the CONFIRM_WK record, which is in the SVM_ITM record:

svm_cepid: ^cell

The address of an identifier that uniquely identifies the connection to SVM and BIP. The identifier is assigned by SVM when the connection is established.

ucepid: ^cell

The address of the user's CEPID.

SV_REJ

This request allows the user to reject a connection request from NAM. This request applies only to A-A connections.

Format

rej_wk: record
svm_cepid: ^cell
text: 0 .. Off(16)

Input

The user sets up the following fields as input in the REJ_WK record, which is in the SVM_ITM record:

svm_cepid: ^cell

The address of an identifier that uniquely identifies the connection to SVM and BIP. The identifier is assigned by SVM when the connection is established.

text: 0 .. Off (16)

This field contains one byte of text explaining why the request was rejected. SVM appends this byte to the NHP header, which is sent to the NOS host.

SV_CALL_T

This request allows the user to request that a terminal connection be established with its peer.

Format

call_t_wk: record
ucepid: ^cell
task: task_ptr
cc_if: ^procedure
bip_if: ^procedure
upline_blk_limit: in + 16
text: buf_ptr

Input

The user sets up the following fields as input in the CALL_T_WK record, which is in the SVM_ITM record:

ucepid: ^cell

The address of user's CEPID.

task: task_ptr

This task identifier is used by SVM to send indications to the user.

cc_if: ^procedure

The address of a procedure used by SVM to inform the user that its request to establish the terminal connection was successful. This indication is sent to the user through a direct call procedure rather than through an ITM. This prevents data from reaching the user before the user has been notified that its connection request has been accepted.

bip_if: ^procedure

The address of a procedure which is used by BIP to send all indications and responses to its users. See chapter 21 in this manual for details.

upline_blk_limit: 0 .. 255

The number of bytes of information that can be sent to the user at any one time. This number is based on the line speed.

text: buf_ptr

A pointer to the text containing 0 through 255 bytes of protocol-related information. SVM appends this text to the header which contains information for the peer user.

SV_CLEAR_T

This request allows the user to cancel a terminal connection.

Format

```
clear_t_wk: record
    svm_cepid: ^cell
    text: buf_ptr
```

Input

The user sets up the following fields as input in the CLEAR_T_WK record, which is in the SVM_ITM record:

svm_cepid: ^cell

The address of an identifier that uniquely identifies the connection to SVM and BIP. The identifier is assigned by SVM when the connection is established.

text: buf_ptr

A pointer to the text explaining why the connection was terminated. SVM appends this text to the header which is sent to the NOS host.

SV_CLEAR

This request allows the user to cancel an application-to-application connection.

Format

clear_wk: record
  svm_cepid: ^cell

Input

The user sets up the following field as input in the CLEAR_WK record, which is in the SVM_ITM record:

svm_cepid: ^cell

The address of an identifier that uniquely identifies the connection to SVM and BIP. The identifier is assigned by SVM when the connection is established.

SV_TERMINAL_CHARACTERISTICS

This service allows the user to request that SVM send the terminal characteristics of a terminal connection to its peer.

Format

tchar_wk: record
  svm_cepid: ^cell
  text: buf_ptr

Input

The user sets up the following fields as input in the TCHAR_WK record, which is in the SVM_ITM record:

svm_cepid: ^cell

The address of the terminal connection whose characteristics are being sent to the peer user.

text: buf_ptr

A pointer to the text containing 0 through 255 bytes of protocol-related information. SVM appends this text to the header which contains information for the peer user.

SV_APPLICATION_ACCOUNTING_STATS

This request allows the user to request that SVM send application statistics to its peer.

Format **aacct_wk: record**
text: buf_ptr

Input The user sets up the following field as input in the AACCT_WK record, which is in the SVM_ITM record:

text: buf_ptr

A pointer to the text containing 0 through 255 bytes of protocol-related information. SVM appends this text to the header which contains information for the peer.

SV_TERMINAL_ACCOUNTING_STATS

This request allows the user to request that SVM send terminal accounting statistics to its peer.

Format

tacct_wk: record
  text: buf_ptr

Input

The user sets up the following field as input in the TACCT_WK record, which is in the SVM_ITM record:

text: buf_ptr

A pointer to the text containing 0 through 255 bytes of protocol-related information. SVM appends this text to the header which contains information for the peer user.

SVM Service Indications

As mentioned earlier, service indications are responses to the service requests sent by SVM users. Service indications are sent through ITMs. A value specified in the workcode parameter of the SV_INDICATION record determines the service being requested. Each workcode value selects a variant of the SV_INDICATION record. Following is the CYBIL definition of the SVM_INDICATION record which is defined in the common deck C7DSVWK.

```
TYPE
  sv_indication = record
    case workcode: sv_reg .. sv_close of

    = sv_reg =
        ind_reg: record
        reglevel: regulation_levels,
        coupler_node: node_range,
      recend,

    = sv_clear =
      ind_clear: record
        ucepid: ^cell,
        text: buf_ptr,
      recend,

    = sv_call_confirm =
      ind_confirm: record
        ucepid: ^cell,
        svm_cepid: ^cell,
        text: buf_ptr,
      recend,

    = sv_call =
      ind_call: record
        svm_cepid: ^cell,
        coupler_node: node_range,
        text: buf_ptr,
      recend,

    = sv_rej =
      ind_rej: record
      ucepid: ^cell,
      source: reject_source,
      text: 0 .. 0ff(16),
      recend,
```

```
  = sv_open =
    ind_open: packed record
      done: boolean,
      coupler_node: node_range,
      mdi_node: node_range,
      np_status: np_status_type,
    recend,

  = sv_close =
    ind_close: packed record
      done: boolean,
      coupler_node: node_range,
    recend,

  casend,
recend;
```

SV_OPEN

This indication is used by SVM to inform the user whether or not its request to open an SVM SAP was successful.

Format

ind_open: packed record
done: boolean
coupler_node: node_range
mdi_node: node_range
np_status: np_status_type

Input

The SVM sets up the following fields as input in the IND_OPEN record, which is in the SV_INDICATION record:

done: boolean

TRUE The request to open an SVM SAP was successful.

FALSE The request to open an SVM SAP was not successful.

coupler_node: node_range

This parameter specifies the node identifier of the NOS host logical link, which is used to support a connection (0 through 255).

mdi_node: node_range

This is the MDI node identifier of the logical link (0 through 255).

np_status: np_status_type

A value (0 through 4) indicating the status of the Network Products interface when the request to open an SVM was made. The user can request a connection only if the NP_STATUS parameter has a value of NP_UP or NP_CONGESTED.

SV_CLOSE

This indication is used by SVM to inform the user whether or not its request to close an SVM SAP was successful.

Format

ind_close: packed record
done: boolean
coupler_node: node_range

Input

The SVM sets up the following fields as input in the IND_CLOSE record, which is in the SV_INDICATION record:

done: boolean

TRUE	The request to close an SVM SAP was successful.
FALSE	The request to close an SVM SAP was not successful.

coupler_node: node_range

This parameter specifies the node identifier of the NOS host logical link, which is used to support a connection (0 through 255).

SV_CALL

This indication is used by SVM to inform the user that its peer has requested a connection with it. This applies only to A-A connections.

Format

ind_call: record
svm_cepid: ^cell
coupler_node: node_range
text: buf_ptr

Input

The SVM sets up the following fields as input in the IND_CALL record, which is in the SV_INDICATION record:

svm_cepid: ^cell

The address of an identifier that uniquely identifies the connection to SVM and BIP. The identifier is assigned by SVM when the connection is established.

coupler_node: node_range

This parameter specifies the node identifier of the NOS host logical link, which is used to support a connection (0 through 255).

text: buf_ptr

A pointer to the text containing 0 through 255 bytes of protocol-related information. SVM strips the NHP header from the data and passes the data on to the user.

SV_CLEAR

This indication is used by SVM to inform the user that the peer user wishes to terminate a connection that was initiated by the user.

Format

ind_clear: record
  ucepid: ^cell
  text: buf_ptr

Input

The SVM sets up the following fields as input in the IND_CLEAR record, which is in the SV_INDICATION record:

ucepid: ^cell

The address of user's CEPID.

text: buf_ptr

A pointer to the text containing 0 through 255 bytes of protocol-related information. SVM strips the NHP header from the data and passes the data on to the user.

SV_CALL_CONFIRM

This indication is used by SVM to inform the user that its request to make an SVM connection was successful. This indication is sent to the user through a direct call interface rather than through an ITM. This is done to prevent data from arriving for the user before the user has been notified that its connection request has been accepted.

Format

ind_confirm: record
ucepid: ^cell
svm_cepid: ^cell
text: buf_ptr

Input

The SVM sets up the following fields as input in the IND_CONFIRM record, which is in the SV_INDICATION record:

ucepid: ^cell

The address of user's CEPID.

svm_cepid: ^cell

The address of an identifier that uniquely identifies the connection to SVM and BIP. The identifier is assigned by SVM when the connection is established.

text: buf_ptr

A pointer to the text containing 0 through 255 bytes of protocol-related information. SVM strips the NHP header from the data and passes the data on to the user.

SV_REJ

This indication is used by SVM to inform the user that its request to make an SVM connection was not successful.

Format

ind_rej: record
ucepid: ^cell
source: reject_source
text: 0 .. Off(16)

Input

The SVM sets up the following fields as input in the SV_REJ record, which is in the SV_INDICATION record:

ucepid: ^cell

The address of the user's CEPID.

source: reject_source

This parameter indicates if the connection request was rejected by SVM or the peer user.

text: 0 .. 0ff (16)

This field contains one byte of data explaining why the connection was rejected.

SV_REG

This indication is used by SVM to inform the user that there has been a change in the status of the Network Products interface.

Format

ind_reg: record
reglevel: regulation_levels
coupler_node: node_range

Input

The SVM sets up the following fields as input in the IND_REG record, which is in the SV_INDICATION record:

reglevel: regulation_levels

A value indicating the status of the network products interface (0 .. 3). One of the following two values is returned to the user:

0 Indicates that all existing connections have been broken and the link is down.

3 Indicates the link is up now.

coupler_node: node_range

This parameter specifies the node identifier of the NOS host whose logical link status has changed (0 through 255).

Constants and Common Types

This section lists the constants and common types used by all the procedures described in this chapter. The common deck C7DSVWK lists all these types and constants.

Constants

```
    svm_registered_title = 'svm_xx'

    first_node = 0,
    last_node = 255,
    min_nhp_cn = 1,
    max_nhp_cn = 255

{Reject reasons returned to NAM in ICN/AP/A

    no_di_resources = 93(16),
    cn_already_in_use = 0c3(16),
    invalid_text = 81(16),
    no_a_to_a_gw = 91(16)

{Reject reasons returned to application in SV_REJECT

    to_appl_no_di_resources = 1,
    to_appl_invalid_text = 2,
    to_appl_link_down = 3

{Regulation level meanings

    regulation_link_down = 0
    regulation_link_congested = 1,
    regulation_link_low_buffers = 2,
    regulation_link_up = 3
```

Common Types

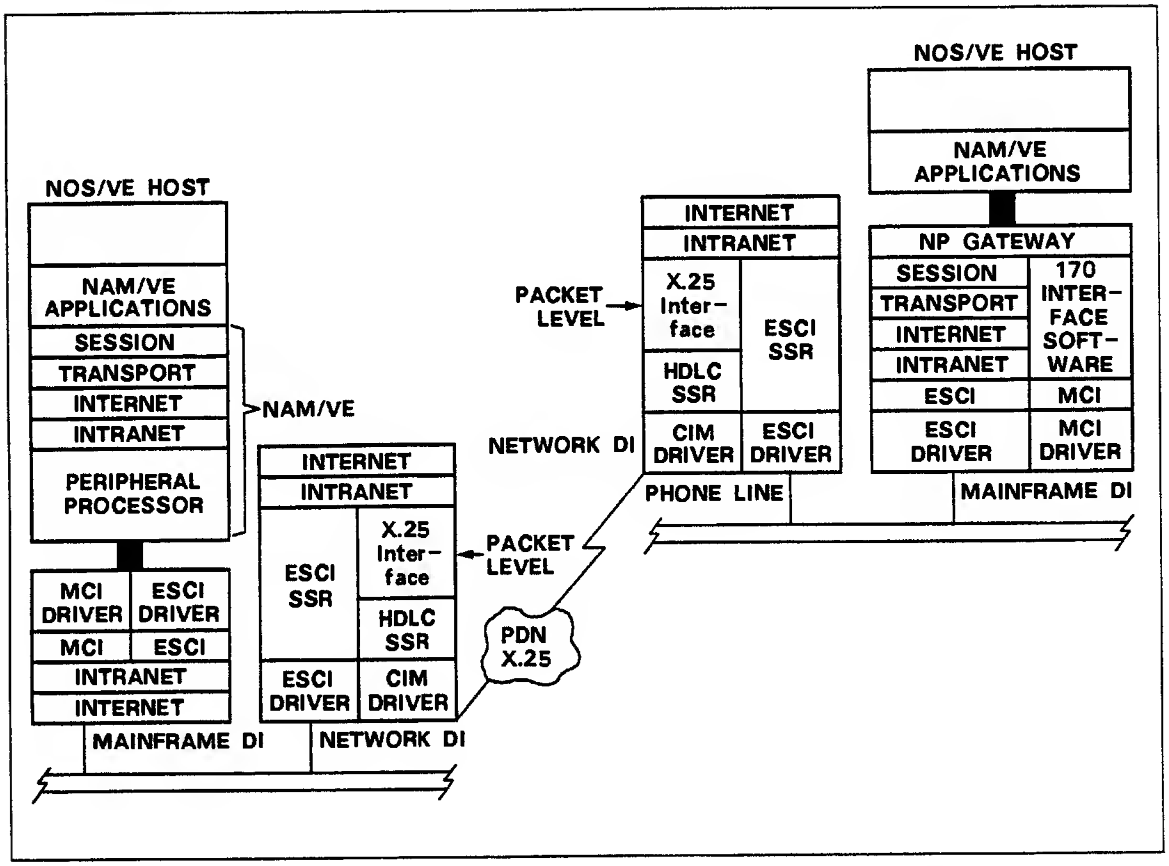
```
cn_range = min_nhp_cn .. max_nhp_cn

node_range = first_node .. last_node

np_status_type =
  np_defined, {NP Interface has been defined
  np_enabled, {NP Interface has been defined and started
  np_up, {NP Interface has been defined, started, and host communication
          has begun
  np_congested, {NP interface is congested
  np_down);  {Communication with the host is down

reg_level = packed record
  pad: 0 .. 7, { reserved, unused
  h: boolean, { host to host
  n: boolean, { NS?
  c: boolean, { CS?
  b: 0 .. 3, { buffer level 0 means logical     link down,
  {3 is up, others currently unused
  pad2: 0 .. 0ff(16), { force goodies to top half of     word
recend

regulation_levels =
    (regulation_link_down .. regulation_link_up)

reject_source = (svm_reject, nam_reject)

shutdown_originator_name = (nam, cp)

svm_workcode = sv_reg .. sv_terminal_accounting_stats

svm_secondary_workcode = ( {
  i_am_fs, { File Server
  i_am_kdisp, {K Display
  i_am_c170gw, {C170 Gateway
  i_am_ivtgw, {IVT Gateway
  i_am_ilog {Independent Log M-E
  )
```

The following workcode definitions for SVM ITMs are listed in the common deck CMDITM.

```
C7 SVM INTERTASK MESSAGE WORKCODE DEFINITIONS (1110-1131)

,sv_reg                          = 1110(16) { Regulation indication
,sv_clear                        = 1111(16) { Clear indication
,sv_call_confirm                 = 1112(16) { Call confirm indication
,sv_call                         = 1113(16) { Call indication
,sv_rej                          = 1114(16) { Reject indication
,sv_open                         = 1115(16) { Open indication
,sv_close                        = 1116(16) { Close indication
,sv_cancel                       = 1117(16) { NP interface request to cancel
,sv_init                         = 1118(16) { Initialization request
,sv_down                         = 1119(16) { Request from NAM
,sv_term                         = 111A(16) { Term request
,sv_call_t                       = 111B(16) { Call request from terminals
,sv_clear_t                      = 111C(16) { Clear request from terminals
,sv_shutdown                     = 111D(16) { Shutdown request
,sv_connection_down              = 111E(16) { Connection down
                                              request from BIP
,sv_terminal_characteristics     = 111F(16) { req. Terminal Char. be sent
                                              to NAM
,sv_application_accounting_stats = 1120(16) { req. appl. acctg. stats sent
                                              to NAM
,sv_terminal_accounting_stats    = 1121(16) { req. term. acctg. stats sent
                                             to NAM
```

(Continued)

- Allows data to be transmitted between two CDCNET systems. As shown in the following figure, the X.25 interface along with the network solution software (Intranet and Internet layers) provides these services.



X.25 Interface between CDCNET Systems

The following chapter describes the Packet Level interface of the X.25 interface. HDLC SSR is described in chapter 20.

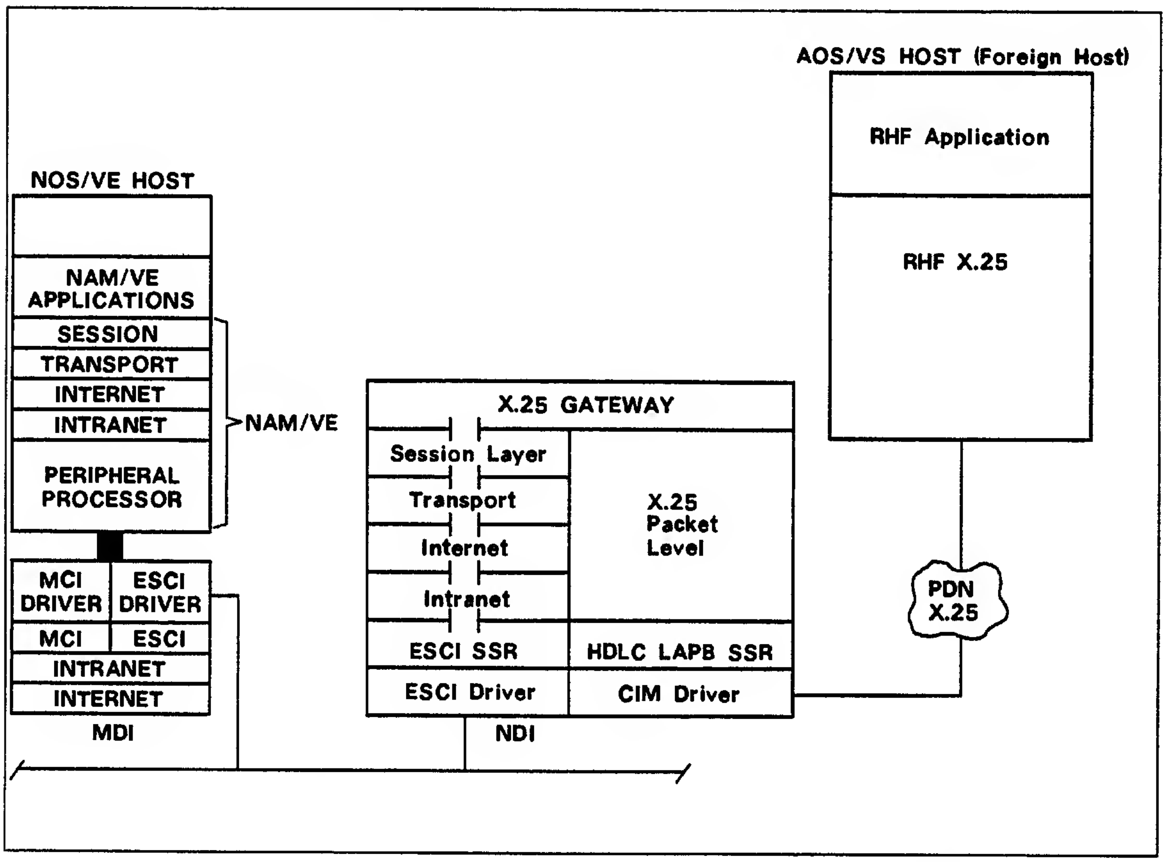
X.25 Interface

The X.25 interface resides in a Network Device Interface (NDI) and consists of the following two software components:

- Packet Level
- HDLC SSR

The X.25 interface provides the following two services:

- Allows applications in any CDCNET system to maintain and use an X.25 virtual circuit to connect to applications in foreign systems accessible across an X.25 PDN or X.25 trunk. As shown in the following figure, the X.25 interface along with the X.25 gateway and the Session layer provides these services.



X.25 Interface between CDCNET and a Foreign Host

(Continued on other side)

X.25 Packet Level 23

X.25 Packet Level 23

This chapter discusses:

- An overview of the CDCNET X.25 Packet Level interface.
- The services required by the X.25 Packet Level interface.
- The services provided by the X.25 Packet Level interface.

This chapter describes only the external interfaces of the CDCNET X.25 Packet Level. It assumes you are familiar with the CCITT Recommendation X.25 and makes no attempts to explain the X.25 concepts and terminology.

Overview

The X.25 Packet Level is part of a three-level architecture defined by the CCITT Recommendation X.25. The X.25 architecture consists of the following layers:

- X.25 Packet Level
- High-Level Data Link Control (HDLC) Link Access Protocol Balanced (LAPB) Link Level
- Physical layer

CDCNET has enhanced the Recommendation X.25 by providing an extra set of services at the X.25 Packet Level. This set of services helps the X.25 interface users, the X.25 gateway, and the X.25 Async TIP perform connection management and data transmission services on virtual circuits.

The CDCNET X.25 Packet Level services thus include the three-level architecture mentioned in Recommendation X.25 and also the extra set of CDCNET X.25 Packet Level services. CDCNET X.25 Packet Level is implemented in a Network Device Interface (NDI) with a direct connection to an X.25 Packet Data Network (PDN) or an X.25 Packet Data Trunk (PDT).

The X.25 Packet Level can be configured, through configuration commands, to support both the DTE and the DCE modes of operations so that foreign systems that only support the DTE mode can also be connected. The CDCNET X.25 Packet Level also supports both the normal (8) and the extended (128) modulo sequencing, switched virtual circuits, and permanent virtual circuits. However, datagram packets, diagnostic packets, and reject packets are not supported by the X.25 Packet Level. The X.25 Packet Level, through SAPs, allows its multiple users to access all or any subset of the underlying HDLC LAPB physical links. See CDCNET Network Configuration and Site Administration Guide for more information on configuration commands.

The types of services provided by the X.25 Packet Level are:

- Layer management services
- Connection management services
- User layer management services
- User connection management services

X.25 Packet Level services are provided by two procedure calls. One is used for layer management services and the other for connection management services. Both these procedures are called with one parameter. This parameter is a variant record which contains a workcode and parameters which are dependent on this workcode. The workcode represents the service being requested or the response to an indication sent earlier by the X.25 Packet Level. The parameters can be either input or output parameters. A return status code parameter conveys the success or failure of the request being processed.

The layer management procedure is an externally referenced (XREF) procedure. The connection management procedure's entry point is returned to the user after the X.25 Packet Level SAP is opened. See Services Provided, later in this chapter, for details.

User layer management services and user connection management services are also provided by two procedures that are called by the X.25 Packet Level to send indications or confirmations.

Each of these procedures is also called with one parameter. This parameter is a variant record which contains a workcode and parameters that are workcode-dependent. The workcode represents the service indications received from the X.25 Packet Level or confirmations of processed requests. The parameters can be either input or output parameters.

The user initiates communication with the X.25 Packet Level by calling the layer management procedure with a PL_OPEN_SAP request. The X.25 Packet Level returns the entry point address of the connection management procedure which the user calls to establish a connection with the its peer and to exchange data. Figure 23-1 illustrates the functional relationship between the X.25 Packet Level and its user.

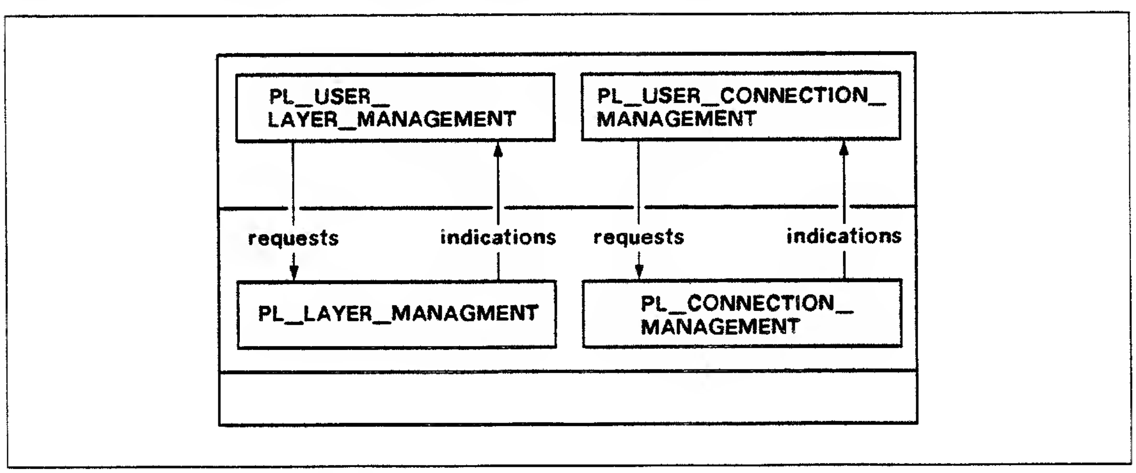


Figure 23-1. X.25 Packet Functional Relationship

Services Required

For the X.25 Packet Level software components to provide services to its own users it, in turn, depends on the services of the other software components. Brief descriptions of the services of each of these software components follow.

HDLC LAPB SSR

The X.25 Packet Level uses the features and services of the HDLC with the LAPB option to transfer and receive data. The X.25 Packet Level and the HDLC LAPB stream service routines (SSR) communicate through procedure calls. See chapter 20 in this manual for more information.

Log ME

The services of the Log ME are used by the X.25 Packet Level to log in abnormal X.25 Packet Level events and statistics. See chapter 8 in this manual for more information.

Services Provided

This section describes the external services provided by the X.25 Packet level to its users. It describes the major data structures used and the service requests exchanged between the X.25 Packet level and its users.

X.25 Packet Level Layer Management Services

The X.25 Packet Level layer management services allow the Packet Level users to perform the preliminary tasks of opening and closing a X.25 Packet Level SAP and establishing a connection.

X.25_LAYER_MGMT_PROC Procedure

Users interface with the X.25 Packet Level through direct calls to the X.25_LAYER_MGMT_PROC procedure. Calls are made with a single parameter which points to a variant record that contains a workcode and parameters which are workcode-dependent. X.25 Packet Level users include the common deck XPXXLMP in their calling modules. Common deck XPXXLMP contains this procedure declaration:

```
PROCEDURE [XREF] x25_layer_mgmt_proc ( {
        request: pl_layer_mgmt_request);
```

The common deck XPXXLMP contains a call to common deck XPDLCMD. XPDLCMD contains the layer management record, the connection management procedure, the user's layer management and connection management procedures, and all the type definitions.

Layer Management Request Record

The layer management request record is the main data structure used by X.25 Packet Level users to open or close an X.25 Packet Level SAP and to establish a connection.

This data structure is a variant record defined with fields to support both input and output parameters. The workcode, service SAP identifier, and status fields are common for all layer management requests, and are defined in the static portion of the layer management record. A value specified in the workcode field determines the other fields or service requests that are to be used in the layer management request.

Following is the CYBIL description of the layer management request record.

```
TYPE
    pl_layer_mgmt_request = record
      workcode: pl_layer_mgmt_codes,
      service_sapid: ^cell,
      status: pl_return_status,
      case pl_layer_mgmt_codes of

      = pl_call_request =
        call_request@: record
         user_cepid: ^cell,
         pl_cepid: ^cell,
         d_bit: boolean,
         address_facil_data: buf_ptr,
         blocksize: 0 .. 0ffff(16),
       recend;

      = pl_open_sap =
        open_sap@: record
         user_sapid: ^cell,
         trunk_names: ^array [1 .. * {<= max_number_ports{ ] of clt$name,
         protocol_ids: ^array [1 .. * {<= max_protocol_ids} ] of 0 .. 255,
         user_layer_mgmt_if: pl_user_layer_mgmt_call,
         user_connection_mgmt_if: pl_user_connection_mgmt_call,
         pl_connection_mgmt_if: pl_connection_mgmt_call,
        recend;

     = close_sap =  no_parameters

      casend,
    recend;
```

Following are the common fields defined in the static portion of the layer management request record:

workcode: pl_layer_mgmt_codes

This parameter must contain one of the following values:

PL_OPEN_SAP

PL_CLOSE_SAP

PL_CALL_REQUEST

The user must supply one of these values to specify a service request when making a layer management request. Details on these service requests are given later in this section.

service_sapid: ^cell

This is the X.25 Packet Level's SAP identifier. This parameter is returned to the user by the X.25 Packet Level. The user specifies this identifier when it sends a request to close a SAP, when it redefines an already opened SAP, and when it sends a request to establish a connection.

status: pl_return_status

This is the status indication of a processed request returned by the X.25 Packet Level. Error codes are listed with each service request later in this chapter. A complete list of status messages with explanations is also given in Constants and Common Types, later in this chapter.

Layer Management Service Requests

The following pages describe the various service requests offered to an X.25 Packet Level user. The user requests a service by specifying an appropriate value in the workcode field of the layer management request record. The common fields described earlier are listed again in the following descriptions of the service requests. The explanation to each common field is not repeated unless there is some unique information for that particular field in a particular service request.

PL_OPEN_SAP

The PL_OPEN_SAP request does the following:

- Allows the user to identify itself to the X.25 Packet Level and to get the address of the X.25 Packet Level's connection management procedure.
- Gets the addresses of the user's layer management and connection management procedures which are used by the X.25 Packet Level to send indications and confirmations to user's requests.

A value of PL_OPEN_SAP in the workcode field of the layer management request initiates this request.

Input The user sets up the following fields as input in the layer management request record:

workcode: pl_layer_mgmt_codes

This field must contain the following value:

PL_OPEN_SAP

user_sapid: ^cell

The user's SAP identifier.

trunk_names: ^array

A pointer to a list of trunk names to which the user would like to establish a connection. A maximum of 32 trunks can be specified.

protocol_ids: ^array

A pointer to a list of protocol identifiers to which the user would like to establish a connection. A maximum of 16 protocol identifiers can be specified.

user_layer_mgmt_if: pl_user_layer_mgmt_call

This is the address of the user's layer management interface which is used by the X.25 Packet Level to send indications and confirmations to layer management requests.

user_connection_mgmt_if: pl_user_connection_mgmt_call

This is the address of the user's connection management interface which is used by the X.25 Packet Level to send indications and confirmations to connection management requests.

Output The X.25 Packet Level returns the following fields as output in the layer management request record:

service_sapid: ^cell

pl_connection_mgmt_if: pl_connection_mgmt_call

This is the address of the X.25 Packet Level's connection management interface to which the user sends all connection management requests and responses.

status: pl_return_status

This is the status indication for the processed request. Following are the status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
pl_no_errors
pl_x.25_trunk_not_defined
pl_duplicate_protocol_id
pl_warning_trunks_not_same_pdn
pl_invalid_open_record
```

PL_CLOSE_SAP

This request allows the user to close a currently open X.25 Packet Level SAP. All established connections associated with this SAP are terminated without notifying the user.

A value of PL_CLOSE_SAP in the workcode field in the layer management request record initiates this request.

Input The user sets up the following fields as input in the layer management request record:

workcode: pl_layer_mgmt_codes

This field must contain the following value:

PL_CLOSE_SAP

service_sapid: ^cell

The X.25 Packet Level's SAP identifier that is being closed.

Output The X.25 Packet Level returns the following field as output in the layer management request record:

status: pl_return_status

This is the status indication for the processed request. Following are the status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
pl_no_errors
pl_unknown_sap
```

PL_CALL_REQUEST

This request allows the X.25 Packet Level user to establish a switched virtual circuit connection with a remote X.25 user. The X.25 Packet Level uses the information passed by the user to generate an X.25 Call Request packet.

A value of PL_CALL_REQUEST in the workcode field of the layer management request record initiates this request.

Input The user sets up the following fields as input in the layer management request record:

workcode: pl_layer_mgmt_codes

This field must contain the following value:

PL_CALL_REQUEST

service_sapid: ^cell

user_cepid: ^cell

The user's CEPID.

d_bit: boolean

The data confirmation bit in the Call Request packet. This field contains one of the following values:

TRUE	The user requests the Packet level to set the delivery confirmation (D) bit in the Call Request packet.
FALSE	The user requests the Packet level not to set the D bit in the Call Request packet.

address_facil_data: buf_ptr

This buffer contains the user's preformatted address, facilities, and call user data. (See CCITT X.25 for information on format.)

The X.25 Packet Level attempts to associate this Call Request packet with a specific X.25 trunk. If the ADDRESS_FACIL_DATA parameter contains a calling DTE address, the X.25 Packet Level looks for an associated X.25 trunk with a matching DTE address field. If the ADDRESS_FACIL_DATA parameter does not contain the calling DTE address, the X.25 Packet Level places the call over the least congested X.25 trunk. The X.25 Packet Level is also responsible for inspecting the ADDRESS_FACIL_DATA parameter for window size and packet size facilities. If present, the Packet level will use the specified values.

NOTE

The user data in a Call Request packet is restricted to a maximum of 16 octets if the fast select facility is not included in the Call Request facilities. The fast select facility allows the user to send a maximum of 128 octets of user data.

blocksize: 0 .. 0ffff (16)

This field indicates the amount of data that the X.25 Packet Level should accumulate before transmitting it to its user. If a zero value is specified, the X.25 Packet Level delivers the data as it is received.

Output The X.25 Packet Level returns the following fields as output in the layer management request record:

pl_cepid: ^cell

The identifier that uniquely identifies the connection to the X.25 Packet Level. All subsequent connection management requests made by the user must include this identifier.

status: pl_return_status

This is the status indication for the processed request. Following are the status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
pl_no_errors
pl_no_trunks_available
pl_no_available_channel
pl_no_memory_available
pl_no_new_connections_allowed
pl_unknown_sap
pl_unknown_cepid
pl_invalid_for_state
pl_trunk_currently_down
```

X.25 Packet Level Connection Management Services

The X.25 Packet Level connection management services allow the Packet level user to manage the activity on connections. Connection management services include:

- Sending both normal and expedited data.
- Resetting and terminating X.25 Packet Level connections.
- Handling flow control.

PL_CONNECTION_MANAGEMENT Procedure

X.25 Packet Level connection management requests are made by calling the PL_CONNECTION_MANAGEMENT procedure. The address of this procedure is passed to the user as an output parameter in a PL_OPEN_SAP request. The connection management request passed in the procedural call is defined below:

```
TYPE
  pl_connection_management_call = ^procedure (
    VAR request: pl_connection_mgmt_request)
```

The X.25 Packet Level Connection Management Request Record

The Packet level connection management request record is the data structure used by X.25 Packet Level users to manage their data exchange. This data structure is a variant record with fields to support both input and output parameters. The workcode, service CEPID, and status fields are common and are used in all connection management requests. A value specified in the workcode field determines other fields or service requests that are to be used in the connection management request.

Following is the CYBIL description of the connection management request record.

```
TYPE

    pl_connection_mgmt_request = record
      workcode: pl_connection_mgmt_codes,
      pl_cepid: ^cell,
      status: pl_return_status,
      case pl_connection_mgmt_codes of

      = pl_call_response =
        call_response@: record
          d_bit: boolean,
          address_facil_data: buf_ptr,
          blocksize: 0 .. 0ffff(16),
        recend

      = pl_clear_request =
        clear_request@: record
          cause: char,
          diag: char,
          address_facil_data: buf_ptr,
          statistics: pl_statistic_rec,
        recend
```

```
  = pl_data_request =
    data_request@: record
     q_bit: boolean,
     d_bit: boolean,
     m_bit: boolean,
     user_data: buf_ptr,
    recend

  = pl_interrupt_request =
    interrupt_request@: record
     data: char,
    recend

  = pl_interrupt_response =
    {no parameters

  = pl_reset_request =
    reset_request@: record
     cause: char,
     diag: char,
    recend

  = pl_reset_response =
      {no parameters

  = pl_data_suppress_request =
      {no parameters

  = pl_data_resume_request =
      {no parameters

  = pl_statistics_request =
    statistic_request@: pl_statistics_rec,

 casend,
recend
```

Following are the common fields defined in the static portion of the Packet level connection management request record:

workcode: pl_connection_mgmt_codes

This field must contain one of the following values:

PL_CALL_RESPONSE

PL_CLEAR_REQUEST

PL_DATA_REQUEST

PL_INTERRUPT_REQUEST

PL_INTERRUPT_RESPONSE

PL_RESET_REQUEST

PL_RESET_RESPONSE

PL_DATA_SUPPRESS_REQUEST

PL_DATA_RESUME_REQUEST

PL_STATISTICS_REQUEST

The user must supply one of these values to specify a service request when making a connection management request. Details on these service requests are given later in this section.

pl_cepid: ^cell

The X.25 Packet Level's CEPID.

status: pl_return_status

This is the status indication of a processed request returned by the X.25 Packet Level. Return codes are listed with each service request later in this chapter. A complete list of status indications with explanations is also given in Constants and Common Types later in this chapter.

The following pages describe the various Packet level connection management service requests offered to an X.25 Packet Level user. The user requests a service by specifying an appropriate value in the workcode field of the connection management request record. The common fields described earlier are listed again in the following descriptions of the service requests. The explanation to each common field is not repeated unless there is some unique information for that particular field in a particular service request.

PL_CALL_RESPONSE

This request allows the user to notify the X.25 Packet Level that it accepts the connection request sent out by its peer entity.

A workcode of PL_CALL_RESPONSE in the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: pl_connection_mgmt_codes

This field must contain the following value:

PL_CALL_RESPONSE

pl_cepid: ^cell

The X.25 Packet Level's CEPID.

d_bit: boolean

The data confirmation bit in the Call Request packet. This field contains one of the following values:

TRUE	The user requests the Packet level to set the delivery confirmation (d) bit in the Call Request packet.
FALSE	The user requests the Packet level not to set the d bit in the Call Request packet.

blocksize: 0 .. 0ffff (16)

This parameter indicates the amount of data that the X.25 Packet Level should accumulate before transmitting it to its user. If a zero value is specified, the X.25 Packet Level delivers the data as it is received.

address_facil_data: buf_ptr

This buffer contains the user's preformatted address and facilities data and the called user's data that the user wishes to include in the Call Accept packet. The X.25 Packet Level inspects the ADDRESS_FACIL_DATA parameter for information on window size and packet size and, if that information is present, the X.25 Packet Level uses the specified values.

NOTE

The called user's data information can be specified only when responding to a CALL INDICATION packet which included the fast select facility.

Output The X.25 Packet Level returns the following field as output in the connection management request record:

status: pl_return_status

This is the status indication for the processed request. Following are the status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
pl_no_errors,
pl_unknown_cepid
pl_invalid_for_state
pl_trunk_currently_down
```

PL_CLEAR_REQUEST

This request enables the user to refuse a connection attempt or to terminate a previously established X.25 Packet Level connection.

A value of PL_CLEAR_REQUEST in the workcode field of the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: pl_connection_mgmt_codes

This field must contain the following value:

PL_CLEAR_REQUEST

pl_cepid: ^cell

cause: char

A value to be placed in the cause code field of the X.25 Clear Request packet.

diag: char

A value to be placed in the diagnostics code field of the X.25 Clear Request packet.

address_facil_data: buf_ptr

This buffer contains the user's preformatted address and facilities and user data that is to be included in the Clear Request Packet.

NOTE

The ADDRESS_FACIL_DATA information can only be specified when the connection was established through a CALL_REQUEST or CALL_INDICATION packet which included the fast select facility.

Output The X.25 Packet Level returns the following fields as output in the connection management request record:

statistics: pl_statistics_rec

The statistics that the X.25 Packet Level accumulated for the connection (table 23-1).

Table 23-1. Statistics Record (PL_STATISTICS_REC)

Field	Content
data_packets_transmitted	The number of data packets transmitted on this virtual circuit. (Type integer.)
data_packets_received	The number of data packets received on this virtual circuit. (Type integer.)
characters_transmitted	The number of characters transmitted on this virtual circuit. (Type integer.)
characters_received	The number of characters received on this virtual circuit. (Type integer.)
segments_transmitted	The number of segments transmitted on this virtual circuit. (Type integer.)
segments_received	The number of segments received on this virtual circuit. (Type integer.)
start_time	The time when the statistics collection started. (Type BCD_TIME.)
end_time	The time when the statistics collection ended. (Type BCD_TIME.)

status: pl_return_status

This is the status indication for the processed request. Following are the status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
pl_unknown_cepid
pl_no_errors
```

PL_DATA_REQUEST

This request allows the X.25 Packet Level user to send data to its peer on an established connection.

A value of PL_DATA_REQUEST in the workcode field of the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: pl_connection_mgmt_codes

This field must contain the following value:

PL_DATA_REQUEST

pl_cepid: ^cell

q_bit: boolean

The qualifier bit is used in data packets to indicate the data type. This field contains one of the following values:

TRUE	The data packet contains control information about the data stream that is be interpreted by a higher-level protocol.
FALSE	The data packet contains unqualified information.

d_bit: boolean

The delivery confirmation (d) bit indicates that the user would like an end-to-end acknowledgment of the data packet being transmitted. This field contains one of the following values:

TRUE	End-to-end acknowlegment of the data packet is requested.
FALSE	Local acknowledgment of the data packet is requested.

m_bit: boolean

The more (m) bit indicates that more data will follow this data packet. This field contains one of the following values:

TRUE	More data will follow this data packet.
FALSE	This data packet completes a packet sequence.

user_data: buf_ptr

The data to be transmitted.

Output The X.25 Packet Level returns the following field as output in the connection management request record:

status: pl_return_status

This is the status indication for the processed request. Following are the status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
pl_no_errors
pl_unknown_cepid
pl_trunk_currently_down
```

PL_INTERRUPT_REQUEST

This request allows the X.25 Packet Level user to send one octet of expedited data over a previously established virtual circuit and bypass the normal data flow control mechanism. There is no confirmation of delivery returned on this request, thus more than one interrupt may be outstanding at any time.

A value of PL_INTERRUPT_REQUEST in the workcode field of the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: pl_connection_mgmt_codes

This field must contain the following value:

PL_INTERRUPT_REQUEST

pl_cepid: ^cell

data: char

This parameter contains one octet of expedited data that is to be transmitted.

Output The X.25 Packet Level returns the following field as output in the connection management request record:

status: pl_return_status

This is the status indication for the processed request. Following are the status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
pl_no_errors
pl_unknown_cepid
pl_trunk_currently_down
```

PL_INTERRUPT_RESPONSE

This request allows the user to acknowledge the receipt of the expedited data.

A value of PL_INTERRUPT_RESPONSE in the workcode field of the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: pl_connection_mgmt_codes

This field must contain the following value:

PL_INTERRUPT_RESPONSE

pl_cepid: ^cell

Output The X.25 Packet Level returns the following field as output in the connection management request record:

status: pl_return_status

This is the status indication for the processed request. Following are the status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
pl_no_errors
pl_unknown_cepid
pl_trunk_currently_down
pl_invalid_for_state
```

PL_RESET_REQUEST

This request allows the X.25 Packet Level user to reinitialize a previously established virtual circuit. The user is not allowed to send any additional reset requests until a RESET_CONFIRM is received from the Packet level.

A value of PL_RESET_REQUEST in the workcode field of the connection management request initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: pl_connection_mgmt_codes

This field must contain the following value:

PL_RESET_REQUEST

pl_cepid: ^cell

cause: char

The information that is to be placed in the cause field of the X.25 Reset Request packet. This parameter gives the peer user the reason(s) for the reset request.

diag: char

The information to be placed in the diagnostics code field of the X.25 Reset Request packet.

Output The X.25 Packet Level returns the following field as output in the connection management request record:

status: pl_return_status

This is the status indication for the processed request. Following are the status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
pl_no_errors
pl_unknown_cepid
pl_trunk_currently_down
pl_invalid_for_state
```

PL_RESET_RESPONSE

This request allows the X.25 Packet Level user to acknowledge the receipt of a reset indication.

A value of PL_RESET_RESPONSE in the workcode field of the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: pl_connection_mgmt_codes

This field must contain the following value:

PL_RESET_RESPONSE

pl_cepid: ^cell

Output The X.25 Packet Level returns the following field as output in the connection management request record:

status: pl_return_status

This is the status indication for the processed request. Following are the status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
pl_no_errors
pl_unknown_cepid
pl_trunk_currently_down
pl_invalid_for_state
```

PL_DATA_SUPPRESS_REQUEST

This request allows the user to request the X.25 Packet Level to exercise flow control on the data. The X.25 Packet Level directly maps this service to an RNR X.25 packet.

A value of PL_DATA_SUPPRESS_REQUEST in the workcode field of the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: pl_connection_mgmt_codes

This field must contain the following value:

PL_DATA_SUPPRESS_REQUEST

pl_cepid: ^cell

Output The X.25 Packet Level returns the following field as output in the connection management request record:

status: pl_return_status

This is the status indication for the processed request. Following are the status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
pl_no_errors
pl_unknown_cepid
```

PL_DATA_RESUME_REQUEST

This request allows the user to request the X.25 Packet Level to stop flow control on the data and start delivering data to the user again.

A value of PL_DATA_RESUME_REQUEST in the workcode field of the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: pl_connection_mgmt_codes

This field must contain the following value:

PL_DATA_RESUME_REQUEST

pl_cepid: ^cell

Output The X.25 Packet Level returns the following field as output in the connection management request record:

status: pl_return_status

This is the status indication for the processed request. Following are the status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
pl_no_errors
pl_unknown_cepid
```

PL_STATISTICS_REQUEST

This request allows the user to request statistics on a connection.

A value of PL_STATISTICS_REQUEST in the workcode field of the connection management request record initiates this request.

Input The user sets up the following fields as input in the connection management request record:

workcode: pl_connection_mgmt_codes

This field must contain the following value:

PL_STATISTICS_REQUEST

pl_cepid: ^cell

Output The X.25 Packet Level returns the following fields as output in the connection management request record:

statistic: pl_statistics_rec

The statistics on the connection (table 23-2).

Table 23-2. Statistics Record (PL_STATISTICS_REC)

Field	Content
data_packets_transmitted	The number of data packets transmitted on this virtual circuit. (Type integer.)
data_packets_received	The number of data packets received on this virtual circuit. (Type integer.)
characters_transmitted	The number of characters transmitted on this virtual circuit. (Type integer.)
characters_received	The number of characters received on this virtual circuit. (Type integer.)
segments_transmitted	The number of segments transmitted on this virtual circuit. (Type integer.)
segments_received	The number of segments received on this virtual circuit. (Type integer.)
start_time	The time when the statistics collection started. (Type BCD_TIME.)
end_time	The time when the statistics collection ended. (Type BCD_TIME.)

status: pl_return_status

This is the status indication for the processed request. Following are the status messages returned for this request. For explanations, see Constants and Common Types later in this chapter.

```
pl_no_errors
pl_unknown_cepid
pl_trunk_currently_down
```

X.25 Packet Level User Layer Management Services

The Packet level user layer management services are provided by the X.25 Packet Level user. The X.25 Packet Level uses these services for the following purposes:

- To give the status of the X.25 Packet Level SAP.
- To deliver call indications to the user.

PL_USER_LAYER_MGMT Procedure

The user layer management indications are sent by the X.25 Packet Level by calling the PL_USER_LAYER_MGMT procedure. The address of this procedure is passed by the user during an OPEN_SAP request. The layer management indication passed in the procedural call to the user is defined below:

```
TYPE
   pl_user_layer_mgmt_call = ^procedure (
      VAR indication: pl_user_layer_mgmt_ind)
```

User Layer Management Indication Record

The PL_USER_LAYER_MGMT_IND is the data structure that is used by the X.25 Packet Level to give the status of the X.25 Packet Level SAP to its users and to indicate that a peer user wishes to establish a connection. This data structure is a variant record with fields to support both input and output parameters. The workcode and the service SAP identifier are common fields and are used in all user layer management indications. A value specified by the X.25 Packet Level in the workcode field determines the other fields or service indications that are to be used in the user layer management indication record.

Following is a CYBIL description of the user layer management indication record:

```
TYPE
  pl_user_layer_mgmt_ind = record
    workcode: pl_user_layer_mgmt_codes,
    user_sapid: ^cell,
    case pl_user_layer_mgmt_codes of

    = pl_sap_status =
      status: pl_sap_status_type,

    = pl_call_indication =
      call_indication: record
      user_cepid: ^cell,
      pl_cepid: ^cell,
      d_bit: boolean,
      address_facil_data: buf_ptr,
      accounting: pl_accounting_rec,
     recend
  casend,
 recend;
```

Following are the common fields defined in the static portion of the user layer management request record:

workcode: pl_user_layer_mgmt_codes

The workcode must contain one of the following values:

PL_SAP_STATUS

PL_CALL_INDICATION

X.25 Packet Level must specify one of these values when sending an indication or confirmation to a layer management request. Details on these service requests are given later in this section.

user_sapid: ^cell

The user's SAP identifier which uniquely identifies the SAP to the user. The user supplies this value when the SAP is first is opened.

User Layer Management Indications

The following pages describe the various indication services offered by the user to the X.25 Packet Level. The X.25 Packet Level requests a service by specifying an appropriate value in the workcode field of the user layer management indication record. The common fields described earlier are listed again in the following descriptions of the service indications. The explanation to each common field is not repeated unless there is some unique information for that particular field in a particular indication.

PL_SAP_STATUS

This indication is used by the X.25 Packet Level to inform the user about the change in the status of the SAP.

A value of PL_SAP_STATUS in the workcode field of the user layer management indication record initiates this indication.

Input The X.25 Packet Level sets up the following fields as input in the user layer management indication record:

workcode: pl_user_layer_mgmt_codes

This field must contain the following value:

PL_SAP_STATUS

This parameter is an ordinal and consists of the following three values:

SAP_DOWN	The X.25 trunks are not available.
SAP_UP	The X.25 trunks are available.
SAP_ CONGESTED	The SAP is congested due to poor memory state or buffer congestion.

user_sapid: ^cell

Output None.

PL_CALL_INDICATION

This indication is used by the X.25 Packet Level to inform the user that a peer X.25 Packet Level user wishes to establish a connection with it.

A value of PL_CALL_INDICATION in the workcode field of the user layer management indication initiates this indication.

Input The X.25 Packet Level sets up the following fields as input in the user layer management indication record:

workcode: pl_user_layer_mgmt_codes

This field must contain the following value:

PL_CALL_INDICATION

user_sapid: ^cell

pl_cepid: ^cell

The X.25 Packet Level's CEPID.

d_bit: boolean

The delivery confirmation (d) bit indicates that the user would like an end-to-end acknowledgment of the data packet being transmitted. This field contains one of the following values:

TRUE	End-to-end acknowlegment of the data packet is requested.
FALSE	Local acknowledgment of the data packet is requested.

address_facil_data: buf_ptr

This parameter contains the formatted DTE and facilities fields and any call user data that came in the INCOMING CALL packet.

accounting: pl_accounting_rec

This parameter includes all the relevent accounting information for the user (table 23-3).

Table 23-3. Accounting Record (PL_ACCOUNTING_REC)

Field	Content
lim_number	The LIM number. (0 .. 255.)
port_number	The port number. (0 .. 255.)
trunk_name:	The trunk name. (Type ALIGNED string 31.)
pdn_id	The name of the public data network .(PDN) or the packet switching network (PSN) that the X.25 Packet Level is interfacing with. (Type ALIGNED string (31).)
logical_channel_number	The logical channel number. (Type ALIGNED 0 .. 0ffff (16).)
send_packet_size	The sending packet size used on the connection. (Type ALIGNED 0 .. 0ffff (16).)
receive_packet_size	The receiving packet size used on the connection. (Type ALIGNED 0 .. 0ffff (16).)
send_window_size	The sending window size used on the connection. (0 .. 255.)
receive_window_size	The receiving window size used on the connection. (0 .. 255.)
calling_throughput_class	This field contains information on the number of bits per second that are being transferred on the connection. (0 .. 255.)
called_throughput_class	This field contains information on the number of bits per second that were transferred on the connection. (0 .. 255.)
call_initiator	The local or remote caller on this connection. (Type ALIGNED string (3).)
charged_system	(ALIGNED string (3).)
dtea	The DTE address. (Type ALIGNED STRING (15).)

Output The user returns the following field as output in the user layer management indication record:

user_cepid: ^cell

The user's CEPID.

X.25 Packet Level User Connection Management Services

The user connection management services are provided by the X.25 Packet Level user. These services are used by the X.25 Packet Level to notify the user that:

- A connection request has been accepted or rejected.
- To indicate there is data for a user from its peer.
- To indicate flow control requests for a connection.

PL_USER_CONNECTION_MGMT Procedure

The user connection management indications are sent by calling the PL_USER_CONNECTION_MGMT procedure. The address of this procedure is passed by the user during an OPEN_SAP request. The connection management indication passed in the procedural call to the user is defined below:

```
TYPE
  pl_user_connection_mgmt_call = ^procedure (
    VAR indication: pl_user_connection_mgmt_ind);
```

X.25 Packet Level User Connection Management Indication Record

The PL_USER_CONNECTION_MGMT_IND is the data structure used by the X.25 Packet Level to send indications related to data exchange.

This data structure is a variant record with fields to support both input and output parameters. The workcode and user CEPID parameters are common and are used in all user connection management indications. A value specified in the workcode field determines the other fields or indications that are to be used in the user connection management indication record.

The following is a CYBIL description of the user layer management indication record:

```
TYPE
  pl_user_connection_mgmt_ind = record
    workcode: pl_user_connection_mgmt_codes,
    user_cepid: ^cell
    case pl_user_connection_mgmt_codes of

    = pl_call_confirm =
      call_confirm@: record
       d_bit: boolean,
       address_facil_data: buf_ptr,
       accounting: pl_accounting_rec,
      recend;

    = pl_clear_indication =
      clear_indication@: record
       clear_origin: pl_clear_origin_types,
       cause: char,
       diag: char,
       address_facil_data: buf_ptr,
       statistics: pl_statistic_rec,
      recend;

    = pl_data_indication =
      data_indication@: record
       q_bit: boolean,
       d_bit: boolean,
       m_bit: boolean,
       user_data: buffer,
      recend;

    = pl_interrupt_indication =
      interrupt_indication@: record
       data: char,
      recend;

    = pl_interrupt_confirm =  {no parameters

    = pl_reset_indication =
      reset_indication@: record
       reset_origin: pl_reset_origin_types,
       cause: char,
       diag: char,
      recend;

    = pl_reset_confirm = {  no parameters

    = pl_data_suppress_indication =  {  no parameters

    = pl_data_resume_indication = {  no parameters

    = pl_statistics_indication =
        statistics_indication@: pl_statistics_rec,

   casend,
  recend;
```

Following are the common fields defined in the static portion of the user connection management indication record:

workcode: pl_user_connection_mgmt_codes

The workcode must contain one of the following values:

PL_CALL_CONFIRM

PL_CLEAR_INDICATION

PL_DATA_INDICATION

PL_INTERRUPT_INDICATION

PL_INTERRUPT_CONFIRM

PL_RESET_INDICATION

PL_RESET_CONFIRM

PL_DATA_SUPPRESS_INDICATION

PL_DATA_RESUME_INDICATION

PL_STATISTICS_INDICATION

The X.25 Packet Level must specify one of these values when sending a user connection management indication. Details on these values are given later in this section.

user_cepid: ^cell

The user's CEPID.

X.25 Packet Level User Connection Management Indications

The following pages describe the various user service requests offered to the X.25 Packet Level. The X.25 Packet Level requests a service by specifying an appropriate value in the workcode field of the user connection management indication record. The common fields described earlier are listed again in the following descriptions of the indications. The explanation to each common field is not repeated unless there is some unique information for that field in a particular indication.

PL_CALL_CONFIRM

This indication allows X.25 Packet Level to inform the user that a peer X.25 Packet Level user has accepted a connection initiated by the user.

A value of PL_CALL_CONFIRM in the workcode field of the user connection management indication record initiates this indication.

Input The X.25 Packet Level sets up the following fields as input in the user connection management indication record:

workcode: pl_user_connection_mgmt_codes

This field must contain the following value:

PL_CALL_CONFIRM

user_cepid: ^cell

d_bit: boolean

The delivery confirmation (d) bit indicates that the user wanted an end-to-end acknowledgment of the data packet transmitted. This field contains one of the following values:

TRUE	End-to-end acknowlegment of the data packet was requested.
FALSE	Local acknowledgment of the data packet was requested.

address_facil_data: buf_ptr

This parameter contains the formatted DTE and facilities fields and any user data that came in the Call Connected packet.

accounting: pl_accounting_rec

This parameter includes all the relevent accounting information for the user. See table 23-3 for more information on this record.

Output None.

PL_CLEAR_INDICATION

This indication allows the X.25 Packet Level to do the following:

- Inform the user that a previously established connection has been terminated.
- Inform the user that a connection request has been rejected by a peer user.

A workcode of PL_CLEAR_INDICATION in the user connection management indication record initiates this indication.

Input The X.25 Packet Level sets up the following fields as input in the user connection management indication record:

workcode: pl_user_connection_mgmt_codes

This field must contain the following value:

PL_CLEAR_INDICATION

user_cepid: ^cell

clear_origin: pl_clear origin_types

This parameter identifies the software component responsible for the terminating the connection. It contains one of the following values:

LOCAL_CLEAR	A clear request was issued by a local system.
REMOTE_CLEAR	A clear request was issued by a remote system.
RESTART_CLEAR	A restart was received on the link.
LINK_FAILURE_ CLEAR	The link failed.
INTERFACE_ STOPPED_CLEAR	The link has been stopped by the operator.

cause: char

This parameter contains one octet of information specifying why the call was cleared. The clearing cause octet is received in the X.25 Clear Indication packet.

diag: char

This parameter contains one octet of the diagnostics code carried in the X.25 Clear Indication packet.

Output None.

PL_DATA_INDICATION

This indication allows the X.25 Packet Level to deliver to the user, on an established connection, the data sent by its peer.

A value of PL_DATA_INDICATION in the workcode field of the user connection management indication record initiates this request.

Input The X.25 Packet Level sets up the following fields as input in the user connection management indication record:

workcode: pl_user_connection_mgmt_codes

This field must contain the following value:

PL_DATA_INDICATION

user_cepid: ^cell

q_bit: boolean

The qualifier bit indicates the data type. This field contains one of the following values:

TRUE	The data packet contains control information about the data stream that is be interpreted by a higher-level protocol.
FALSE	The data packet contains unqualified information.

d_bit: boolean

The delivery confirmation (d) bit indicates that the user wanted an end-to-end acknowledgment of the transmitted data packet. This field contains one of the following values:

TRUE	End-to-end acknowlegment of the data packet was requested.
FALSE	Local acknowledgment of the data packet was requested.

m_bit:

The more (m) bit indicates that more data will follow this data packet. This field contains one of the following values:

TRUE	More data will follow this data packet.
FALSE	This data packet completes a packet sequence.

address_facil_data: buf_ptr

The data to be delivered to the user.

Output None.

PL_INTERRUPT_INDICATION

This indication allows the X.25 Packet Level to inform the user that the peer user has bypassed normal data on a connection. The user should respond to this indication with an INTERRUPT_RESPONSE request.

A value of PL_INTERRUPT_INDICATION in the workcode field of the user connection management indication record initiates this indication.

Input The X.25 Packet Level sets up the following fields as input in the user connection management indication record:

workcode: pl_user_connection_mgmt_codes

This contains the following value:

PL_INTERRUPT_INDICATION

user_cepid: ^cell

data: char

This field contains one byte of expedited data.

Output None.

PL_INTERRUPT_CONFIRM

This indication allows the X.25 Packet Level to inform the user that its peer has acknowledged the receipt of the expedited data. The user can generate INTERRUPT_REQUEST requests again.

A value of PL_INTERRUPT_CONFIRM in the workcode field of the user connection management indication record initiates this indication.

Input The X.25 Packet Level sets up the following fields as input in the user connection management indication record:

workcode: pl_user_connection_mgmt_codes

This field must contain the following value:

PL_INTERRUPT_CONFIRM

user_cepid: ^cell

Output None.

PL_RESET_INDICATION

This indication allows the X.25 Packet Level to inform the user that a peer user in a remote system or the Packet level has requested to reset the virtual circuit. The user responds to this request with a RESET_RESPONSE request.

A value of PL_RESET_INDICATION in the workcode field of the user connection management indication record initiates this indication.

Input The X.25 Packet Level sets up the following fields as input in the user connection management indication record:

workcode: pl_user_connection_mgmt_codes

This field must contain the following value:

PL_RESET_INDICATION

user_cepid: ^cell

reset_origin: pl_reset_origin_types

This field indicates the origin of the reset request and consists of the following two values:

LOCAL_RESET	The reset request was made by a local system.
REMOTE_RESET	The reset request was made by a remote system.

Output None.

PL_RESET_CONFIRM

This indication allows the X.25 Packet Level to inform the user that the reset request it had made earlier is complete. The user can now generate RESET_REQUESTS again. A value of PL_RESET_CONFIRM in the workcode field of the user connection management indication record initiates this indication.

Input The X.25 Packet Level sets up the following fields as input in the user connection management indication record:

workcode: pl_user_connection_mgmt_codes

This field must contain the following value:

PL_RESET_CONFIRM

user_cepid: ^cell

Output None.

PL_DATA_SUPPRESS_INDICATION

This indication is issued by the X.25 Packet Level to inform the user to stop sending data on the connection.

A value of PL_DATA_SUPPRESS_INDICATION in the workcode field of the user connection management indication record initiates this indication.

Input The X.25 Packet Level sets up the following fields as input in the user connection management indication record:

workcode: pl_user_connection_mgmt_codes

This field must contain the following value:

PL_DATA_SUPPRESS_INDICATION

user_cepid: ^cell

Output None.

PL_DATA_RESUME_INDICATION

This indication allows the X.25 Packet Level to notify the user that it can now start issuing data requests again.

A value of PL_DATA_RESUME_INDICATION in the workcode field of the user connection management indication record initiates this indication.

Input The X.25 Packet Level sets up the following fields as input in the user connection management indication record:

workcode: pl_user_connection_mgmt_codes

This field must contain the following value:

PL_DATA_RESUME_INDICATION

user_cepid: ^cell

Output None.

PL_STATISTICS_INDICATION

This indication allows the X.25 Packet Level to deliver statistics for a particular connection.

A value of PL_STATISTICS_INDICATION in the workcode field of the user connection management indication record initiates this indication.

Input The X.25 Packet Level sets up the following fields as input in the user connection management indication record:

workcode: pl_user_connection_mgmt_codes

This field must contain the following value:

PL_STATISTICS_INDICATION

user_cepid: ^cell

statistics: pl_statistic_rec

The statistics accumulated for the connection. See table 23-2 for details on this record.

Output None.

Constants and Common Types

This section lists the constants and common types used by all the procedures described in this chapter. The common deck XPDLCMD lists all these types and constants.

Constants

```
max_dte_size = 15,
max_facility_size = 63,
max_protocol_ids = 16,
max_user_data_non_fast_select = 16,
max_user_data_fast_select = 128

pl_call_request = 1,
pl_open_sap = 2,
pl_change_sap = 3,
pl_start_pvc = 4,
pl_close_sap = 5,
last_pl_layer_mgmt_code = pl_close_sap

pl_first_user_request = pl_call_response,
pl_call_response = 1,
pl_clear_request = 2,
pl_data_request = 3,
pl_interrupt_request = 4,
pl_interrupt_response = 5,
pl_reset_request = 6,
pl_reset_response = 7,
pl_data_suppress_request = 8,
pl_data_resume_request = 9,
pl_stop_pvc_request = 10,
pl_statistics_request = 11,
pl_last_user_request = pl_statistics_request
```

Common Types

```
pl_call_origin_types = (local_call, remote_call)

pl_clear_origin_types = (local_clear, remote_clear, restart_clear,
                         link_failure_clear, interface_stopped_clear)

pl_connection_mgmt_call = ^procedure (
  VAR request: pl_connection_mgmt_request)

pl_connection_mgmt_codes = pl_call_response .. pl_last_user_request

pl_layer_mgmt_call = ^procedure (
  VAR request: pl_layer_mgmt_request)

pl_layer_mgmt_codes = pl_call_request .. last_pl_layer_mgmt_code

pl_reset_origin_types = (local_reset, remote_reset)

pl_sap_status_type = (pl_sap_down, pl_sap_congested, pl_sap_up)
```

```
pl_user_connection_mgmt_call = ^procedure (
  VAR indication: pl_user_connection_mgmt_ind)

pl_user_connection_mgmt_codes = (
  pl_call_confirm,
  pl_clear_indication, { graceful close or response to same
  pl_data_indication, { incoming normal data
  pl_interrupt_indication, { incoming expedited data
  pl_interrupt_confirm, { acknowledgment
  pl_reset_indication, { reset of connection
  pl_reset_confirm, { acknowledgment
  pl_data_suppress_indication, { stop data flow
  pl_data_resume_indication, { resume data flow
  pl_statistics_indication); { statistics record
 pl_user_layer_mgmt_call = ^procedure (
  VAR indication: pl_user_layer_mgmt_ind)

pl_user_layer_mgmt_call = ^procedure (
  VAR indication: pl_user_layer_mgmt_ind)

pl_user_layer_mgmt_codes = (
  pl_sap_status,
  pl_call_indication)

pl_return_status = (pl_no_errors,
  pl_x25_trunk_not_defined, {pl_open_sap/pl_change_sap/pl_start_pvc
  pl_duplicate_protocol_id, {pl_open_sap/pl_change_sap
  pl_warning_trunks_not_same_pdn, {pl_open_sap/pl_change_sap
  pl_invalid_open_record, {pl_open_sap
  pl_unknown_sap,{pl_change_request/pl_closed_sap/pl_start_pvc/pl_close_sap
  pl_no_trunks_available, {pl_call_request
  pl_no_available_channel, {pl_call_request
  pl_no_memory_available, {pl_call_request/pl_start_pvc
  pl_no_new_connections_allowed, {pl_call_request/pl_start_pvc
  pl_pvc_channel_not_defined, {pl_start_pvc
  pl_pvc_already_assigned, {pl_start_pvc
  pl_unknown_cepid, {all connection requests
  pl_invalid_for_state, {all connection requests
  pl_trunk_currently_down {all connection requests
  )
```

pl_no_errors	The request was processed without any errors.
pl_x25_trunk_not_defined	The trunk name specified in the request has not been defined.
pl_duplicate_protocol_id	The X.25 Packet Level has another SAP opened for the same trunk name with the same protocol identifier. Protocol identifiers must be unique for each X.25 trunk name.
pl_warning_trunks_not_same_pdn	The trunk names specified in the request are not all of the same Packet Data Network Type.
pl_invalid_open_record	The trunk names and protocol identifier specified in the PL_OPEN_SAP request exceed the allowed value.
pl_no_trunks_available	All trunk names associated with this SAP identifier are currently down, or the calling DTE address field is directed at an X.25 trunk that is currently down.
pl_no_available_channel	There are no available logical channels on the trunk names associated with this SAP identifier.
pl_no_memory_available	System memory resources are not available.
pl_no_new_connections_allowed	No new connections are being allowed because of congested or poor memory status.
pl_pvc_channel_not_defined	The X.25 Packet Level is not configured to support permanent virtual circuits.
pl_pvc_already_assigned	The PVC channel has already been assigned to another X.25 Packet Level user.
pl_unknown_cepid	The CEPID is not recognized as a valid X.25 Packet Level identifier.
pl_invalid_for_state	User has made an out-of-sequence service request.
pl_trunk_currently_down	Attempting to establish a connection on a trunk that is currently not available.

Appendixes

Glossary A

A

Alarm

A message displayed at a CDCNET network operator console/terminal; it describes an operational event that affects a CDCNET device interface or its related network resources.

Alarm Management Entity

The software component that displays alarm messages received from all CDCNET software components.

Application Layer

Layer 7 of the OSI architecture (not described in detail by the ISO). This layer provides services that directly support user and application tasks and provides system management functions.

Application-to-Application

An end-to-end link between an application layer of one system and the application layer of another for the exchange of information.

B

Base System Software

CDCNET software that works to initialize the device interface and maintain it during operation. Base System Software also provides a set of common routines for use by other CDCNET software.

Batch Transfer Service (BTS)

Control Data's terminal-to-application protocol that enables batch terminals to perform remote job entry with hosts executing NOS or NOS/VE.

BIP

Refer to Block Interface Program.

Block

In the context of network communications, a portion or all of a message. A message is divided into blocks to facilitate buffering, transmission, error detection, and correction for variable-length data streams. Differing block protocols apply to the host-to-device interface and the device-interface-to- terminal-interfaces.

During input from a terminal, a block is a single transmission consisting of one or more lines of one or more messages.

During input to a service, a block is a single line consisting of part or all of a message. Terminal transmission blocks are divided into as many service input blocks as needed, until the message is completed.

During output from a host application program, a block is one or more lines. During output from a device interface to a terminal, a block is one terminal transmission buffer.

Block Interface Program (BIP)

Software that, along with other interfaces and gateways, connects a NOS host to CDCNET. BIP provides services for data transfer across an established Network Products connection. Refer also to Service Module.

Boot File

A file that contains the basic set of software that is loaded into a device when the device interface requests to be loaded. A boot file brings a device interface up to a basic operational state. Further definition of the device interface's functions is provided by its system configuration file.

BTS

Refer to Batch Transfer Service.

Byte

A group of contiguous bits. Unless prefixed (for example, a 6-bit byte), the term implies 8-bit groups. An 8-bit byte is sometimes called an octet. When used for encoding character data, a byte represents a single character.

C

Catenet

Refer to Concatenated Network.

A group of connected CDCNET network solutions. This term is often used when referring to all the device interfaces and network solutions in a site's network.

CCITT

Refer to Consultative Committee of International Telephone and Telegraph.

CDCNET

Refer to Control Data Distributed Communications Network.

CDCNET System

Refers to a device interface.

CDNA

Refer to Control Data Network Architecture.

CEP

Refer to Connection Endpoint.

CEPID

Refer to Connection Endpoint Identifier.

Certified Release

A release of CDC software that includes host operating system software and the network used with the operating system. An example of a certified release is a version of NOS plus CDCNET software.

Channel Capacity

The maximum baud rate that can be handled by a channel.

Clock Management Entity

The software component that manages and synchronizes a real-time clock and calendar for every CDCNET device interface.

Command Management Entity

The software component that receives and generates responses to commands, such as configuration, diagnostics, status, and statistics commands.

Communications Interface Module (CIM)

The logic board within a CDCNET device interface that controls transmissions between the Line Interface Module (LIM) bus and the internal system bus (ISB).

Concatenated Network

A communications network composed of more than one type of communications medium (more than one network solution); often established when it is necessary to interconnect a local area network with other resources (for example, another local area network or geographically remote computer-related resources). Also called a catenet.

Configuration Command

A command that establishes, cancels, or redefines the configuration of a network component in the network's logical definition.

Configuration Procurer

Software that obtains and submits for execution the device interface configuration file commands to the Command ME. This is done at system startup.

Confirm Primitive

Service that allows the current layer to inform the adjacent higher layer that a requested service has been completed.

Congestion

A condition in which there is more message traffic on a network solution or communication line than the line's carrying capacity. Continued congestion results in lengthy message delay and discarding of new messages.

Connection Endpoint (CEP)

Defined within a SAP to establish a unique connection between two entities.

Connection Endpoint Identifier (CEPID)

An identifier used to distinguish different connections associated with a single SAP.

Consultative Committee of International Telephone and Telegraph (CCITT)

An organization chartered by the United Nations to develop and publish international standards for the communications industry.

Control Data Distributed Communications Network (CDCNET)

1. The collection of compatible hardware and software products offered by Control Data Corporation to interconnect computer resources into distributed communications networks.

2. A network that is interconnected by Control Data Network Architecture (CDNA)-compatible hardware and software products.

Control Data Network Architecture (CDNA)

The network architecture designed by Control Data Corporation. CDNA follows the lower layer recommendations of the International Standards Organization's (ISO) Open System Interconnection (OSI) reference model.

Cost

A relative measure assigned to a path (such as a network solution) that is used for transmitting data through a CDCNET-type network. The cost of each possible path is computed and stored into tables by the Routing Management Entity (ME). From these tables, the Routing ME determines the path that has the least cost. The path with the least cost is used to transmit data. The cost of a path may change depending upon the amount of congestion on the path. A congested network solution has a higher cost than an uncongested network solution.

Coupler

A hardware module on a Mainframe Device Interface (MDI) that connects a host's peripheral processor to CDCNET.

CYBIL

Primary implementation language for NOS/VE computer systems and CDCNET software.

D

Data Buffer

A structure for storing user data in device interface memory; contrast with descriptor buffer. A pointer is associated with the first character of data in the buffer. Data buffer length is configurable.

Data Communication

The interchange of data messages from one point to another over communications channels.

Data Communications Equipment (DCE)

The hardware that links data terminal equipment (DTE) to communications media. Data communications equipment is normally a modem or modem equivalent (data set).

Data Link

An error-checked communications pathway between device interfaces over a physical medium. Data link protocol frames messages for transmission. The integrity of received messages is checked. Data link manages access to and use of the medium, and ensures proper sequencing of transmitted data.

Data Terminating Equipment (DTE)

Data communications equipment that allows human interaction with the databases and operations of a network.

Data Transmission

The passing of information over physical media from source to receiver.

Data Unit

Data that is transmitted as a unit between peer entities in different systems. Refer also to Protocol Data Unit (PDU).

Datagram

A packet switching service in which packets are independently routed.

DCNS

Refer to Distributed Communications Network Software.

Default

A preselected value supplied for a missing parameter upon the entry of a command or subcommand.

Dependent Log ME

Refer to Log Management Entity.

Descriptor Buffer

A data structure used for chaining data buffers.

Device Interface (DI)

The communications processor that Control Data offers as its CDCNET hardware product. Also called a CDCNET device interface.

Device Manager (DVM)

A set of routines responsible for the interface between CDNA's physical and link layers (layers 1 and 2, respectively).

DI

Refer to Device Interface.

Directly Connected Network Solution

A network solution is said to be directly connected to a system if that system can directly transfer data to that network solution.

Directory Management Entity

A software component that maintains titles and addresses that identify the individual software components in a CDCNET network.

Distributed Communications Network Software (DCNS)

The software that executes in a device interface as part of the CDCNET product.

Down

A status of suspended service.

DTE

Refer to Data Terminating Equipment.

DVM

Refer to Device Manager.

E

Echo ME

The software component that verifies a particular system in the catenet is operational by returning a message to the user in the system from which the message was received.

Entity

Software function or group of software functions residing within a Control Data Network Architecture (CDNA) layer.

Error Management Entity

The software component that generates an Internet error report (IER) for a message in error and sends the IER to the message source. If it is not appropriate to send an error report back to the message source, the Error ME logs the error message.

ESCI

Refer to Ethernet Serial Channel Interface.

Ethernet Serial Channel Interface (ESCI)

The logic board within a CDCNET device interface that controls transmissions between an Ethernet transceiver and the internal system bus (ISB) of the device interface.

Executive

A realtime monitor for the device interface which acts as the center of its software operating system. The Executive implements a set of procedures that enable users to efficiently share the system's available processing power and memory resources. It also provides some performance information about its own operation and that of its users.

A part of a device interface's base system software that enables the device interface to perform such operating system functions as CPU scheduling, memory and buffer management, timer services, intertask communications, exception processing, and interrupt vector services.

F

Failure Management

The set of software procedures implemented by Control Data Network Architecture (CDNA) for detection of, and recovery from, hardware or software failures in a CDCNET system.

FIFO

Refer to First-In/First-Out.

File Access Management Entity

The software component that accesses and manipulates permanent files residing on host computers configured within a CDCNET network.

File Control Block

Data structure providing the user interface to the Dependent File Access ME.

First-In/First-Out (FIFO)

This term applies to data processing services in which requests are serviced in the same order they are received.

Frame

A group of bits that includes the message and address information. A frame contains a block of data. The frame is the basic communication unit used in device-interface-to-device-interface communications, and provides high data density over data-grade lines, as well as data assurance.

G

Gateway

A software interface between systems with different architectures and protocols.

Gateway Title

The logical title assigned to a gateway during network definition.

H

Hardware

Electronic circuitry and its housing, including cabinetry, power hookup, and cooling system.

HDLC

Refer to High-Level Data Link Control.

Header

The portions of a block holding information about the block source, destination, type, and/or priority. During network movement, a block can acquire and discard several headers. Headers are discarded at the appropriate stage of processing. Also called a heading or leader.

High-Level Data Link Control (HDLC)

The International Standards Organization's (ISO) bit-oriented protocol for the data link layer of the Open Systems Interconnection (OSI) reference model.

Hop

A measurement representing the distance between two or more locally connected network solutions. Each hop represents the crossing of a single boundary separating two network solutions.

Host

Refer to Host Computer.

Host Computer

A mainframe computer system, connected to a communications network, that provides primary services such as data base access, user application execution, or program compilation. For CDCNET, a host computer provides network support functions, including maintenance of device interface load files.

I

IER

Refer to Internet Error Reports.

Indication Primitive

Service that allows the current (N) layer to indicate to the adjacent higher layer (N + 1) that the layer received a request to activate a particular service from a peer.

Initialization Management Entity

A management entity (ME) that loads, dumps, and initializes device interface software components.

Initialization ME

A CDCNET software management entity (ME) that controls the loading and dumping of device interfaces. The Dependent Initialization ME resides in every device interface and is responsible for requesting that the device interface be loaded by the host or by other fully-loaded device interfaces. The Independent Initialization ME resides in a fully-loaded device interface and can manage the loading of another device interface that requests to be loaded.

Interface

A mechanism that enables the exchange of data between two dissimilar resources in a communications network.

International Standards Organization (ISO)

A worldwide standards group similar in function to the American National Standards Institute (ANSI). ANSI is a member of International Standards Organization.

Internet Error Report (IER)

Mechanism used by Error Management Entity to report detected errors. An Internet error report contains an Internet Protocol Data Unit, created by Error ME, containing the error number, error parameter, and the 3B SAP identifier of the software component that detected the error.

Intertask Message (ITM)

A mechanism for delivering information (messages) from one task to another task within the system.

Intranet Layer

Layer 3A of Control Data Network Architecture. This layer provides the interface between individual network solutions software (layers 1 and 2) and higher-layer CDNA software.

ISO

Refer to International Standards Organization.

ITM

Refer to Intertask Message.

K

K Display

A NOS host console display that enables operators to interact with various operating system utilities (for example, those controlling user validation and NAM subsystem interaction).

L

LAN

Refer to Local Area Network.

Layer

A collection of related network processing functions with specified interfaces to other similarly collected functions. Modern data communications architectures are defined using a hierarchical set of layers.

Link

1. Any specified relationship between two device interfaces in a network, or a communication path between two device interfaces, or a data link.
2. The communications path between two device interfaces. Also called a line, channel, or circuit.

Local Area Network (LAN)

A privately owned communications network that interconnects computer-related resources. Typically, the resources interconnected by this network are confined to a relatively concise geographic area (such as a single building).

Log File

A file that is created and maintained by the operating system for storing error information and usage data concerning system elements.

Log Group

Group or subset of device interfaces within the catenet that send their log messages to a common log file.

Log Management Entity (Log ME)

Software that manages the transmission and recording of log messages generated by device interface software. Consists of Dependent and Independent Log Management entities. Dependent Log Management Entities, residing in device interfaces, are sources of log messages. Independent Log Management Entities, residing in a host-connected device interface, work with host applications or a NOS/VE host to write the log messages to the network's log file on the host.

Log Support Application (LSA)

Also known as the Dependent Log Management Entity and/or source logging function. Software that manages the generation and transmission of log messages generated by device interface software. Resident in every device interface.

Logging

The process of issuing messages for network activity and recording the messages in a log file.

LSA

Refer to Log Support Application.

M

Main Processor Board (MPB)

The logic board within a CDCNET device interface that provides the primary processing power for the device interface.

Mainframe Channel Interface (MCI)

An optional logic board within a CDCNET device interface that connects the device interface to a 12-bit CYBER host channel.

Mainframe Device Interface (MDI)

The standard CDCNET device interface variant that interconnects host computers operating under NOS or NOS/VE with an Ethernet local area network.

Mainframe Terminal Device Interface (MTI)

The standard CDCNET device interface variant that interconnects NOS and NOS/VE host computers with terminals, workstations, and unit record equipment without requiring a local area network.

Maintenance Software

Software designed to perform system tests and diagnostics. All CDCNET maintenance software is onboard and online.

Management Entity (ME)

CDCNET software that performs network management functions. CDCNET supports various MEs to perform specific network tasks.

MCI

Refer to Mainframe Channel Interface.

MDI

Refer to Mainframe Device Interface.

ME

Refer to Management Entity.

Message Template

String of text characters and control sequences used to generate a log message or command response that can be displayed at an operator console.

MPB

Refer to Main Processor Board.

MTI

Refer to Mainframe Terminal Device Interface.

Multicast Address

A single address through which a set of systems, connected to the same network solution, can be grouped together and identified.

N

NAM

Refer to Network Access Method.

NAM/VE

Refer to Network Access Method/Virtual Environment.

NDI

Refer to Network Device Interface.

Network Access Method (NAM)

The access method that resides under NOS; allows host-based network applications programs to exchange information with communications networks.

Network Access Method/Virtual Environment (NAM/VE)

The access method that resides under NOS/VE; allows host-based network applications programs to exchange information with communications networks.

Network Address

String of ASCII characters that identify a particular network. Network addresses contain the network identifier, system identifier, and SAP identifier.

Network Architecture

A set of functional layers in which each layer performs a specific set of functions and services; together, the layers interact to provide total, end-to-end network operation. Each layer uses a protocol and has its relationship with other layers defined.

Network Control Layer

The third layer in the International Standard Organization architecture. Controls the flow of messages between nodes by addressing messages, setting up the path between communication nodes, and routing messages across nodes.

Network Device Interface (NDI)

The standard CDCNET device interface variant that transfers data between networks (for example, between two local area networks, between a local area network and a communications line, or between a local area network and a public data network).

Network Host Products (NHP)

Refer to Network Products.

Network Identifier

A unique 32-bit character string that identifies a particular network solution.

An identifier that specifies the network on which a system is located. Each network solution within a catenet is assigned a unique 32-bit network identifier.

Network Operating System (NOS)

The operating system for CYBER 170 computer systems.

Network Operating System/Virtual Environment (NOS/VE)

The software that controls data processing and storage in CYBER 180 mainframes. CDCNET files stored and processed in CYBER 180 mainframes, such as configuration and boot files, network log files, and CDCNET host applications are run under the Network Operating System/Virtual Environment.

The virtual operating system for CYBER 180 computer systems.

Network Operator

A person who monitors CDCNET activity, has the ability to control CDCNET hardware and software, makes occasional network configuration changes, and performs elementary troubleshooting by sending commands to the network's device interfaces. A network operator may perform these tasks from a host console or a remote terminal.

Network Performance Analyzer (NPA)

The CDCNET software utility that generates statistical reports based on its analysis of the network log file or generates event/error reports based on log messages in the network log file.

Network Products (NP)

Programs that run under NOS in a host mainframe to allow data and computer applications to be transmitted from the mainframe through a computer network. Network Products include Network Access Method (NAM) and Network Definition Language (NDL). Network Products and CDCNET have different architectures. For hosts to send data through CDCNET, the Mainframe Device Interfaces connected to the mainframes must have gateways to translate between Network Products and CDCNET protocols. Also referred to as Network Host Products.

Network Solution

A communications medium over which data is transmitted between interconnected network resources. A network solution differs from other communications lines because it is shared by multiple network resources (it is not solely dedicated to the handling of data transmissions between a single pair of network resources).

Represents a combination of a physical medium and layers 1, 2, and 3A which together interconnect two or more CDCNET device interfaces.

NHP

Refer to Network Host Products.

NP Gateway

Network Products Gateway. A program that runs in a Mainframe Device Interface (MDI) or Mainframe Terminal Device Interface (MTI) connected to a host mainframe, and which allows the host to send applications and data through a CDCNET to other mainframes. The gateway acts as a protocol converter between Network Products and CDCNET.

NP IVT Gateway

Network Products Interactive Virtual Terminal Gateway. A program which runs in a Mainframe Device Interface (MDI) or Mainframe Terminal Device Interface (MTI) connected to a host mainframe, and which allows the host mainframe to send applications through CDCNET to interactive terminals. The gateway acts as a protocol converter between the host's Network Products protocols and CDCNET protocols.

NPA

Refer to Network Performance Analyzer.

O

Octet

An 8-bit byte.

Online Loader

A CDCNET service that loads software into device interfaces when the software is needed while the network is operational, as opposed to initial loader, which loads software into device interfaces only when they are started up (initialized).

Open System Interconnection (OSI)

The International Standards Organization's reference model for network processing. This model is based on a network architecture that segregates network functions into seven layers.

OSI

Refer to Open System Interconnection.

P

Packet

A group of binary digits, including data and control elements, switched and transmitted as a data unit by communications networks. The packet's data, control signals, and error-control information are arranged in a specific format. Different types of networks use different sizes of packets.

Packet-Switching

1. A data transmission process using addressed packets whereby a channel is occupied only for the duration of transmission of the packet.
2. The process by which data packets are placed on the channel and travel to the destination.
3. A method of transmitting messages through a communications network in which long messages are subdivided into short packets with a maximum length.

Parity Check

An error detection method in which an extra bit is added to data to make the number of 1s in each grouping of bits either always odd (for odd parity), or always even (for even parity).

PDU

Refer to Protocol Data Unit.

Peer Protocol

The rules by which entities in the same layer convey the neccessary control information to support communication.

Physical Link

Layer 1 of the Open System Interconnection model. Defines the electrical and mechanical aspects of interfacing to a physical medium for transmitting data, as well as setting up, maintaining, and disconnecting physical links. Includes a software device driver for each communications device, and the hardware (interface devices, modems and communications lines).

Point-to-Point Connection

A network configuration in which a connection is established between two device interfaces.

Presentation Layer

Layer 6 of the ISO architecture. Determines the format and visual presentation of displayed data.

Primitive

Interface between entities in adjacent layers. Each primitive provides a service to an entity.

Protocol

A set of conventions that must be followed to achieve complete communications between the computer-related resources in a network. A protocol can reflect the following:

1. A set of predefined coding sequences, such as the control byte envelopes added to (or removed from) data exchanged with a terminal.
2. A set of data addressing and division methods, such as the block mechanism used between a network application program and Network Access Method.
3. A set of procedures that control communications, such as the supervisory message sequences used between a network application program and Network Access. Method.

Protocol Data Unit (PDU)

A data unit that is used to communicate information between peer entities.

Public Data Network

A commercial packet-switching network that supports the communications interface described in CCITT protocol X.25.

R

RCB

Refer to Registration Control Block.

Real Time Clock (RTC)

A time-of-day clock maintained by the device interface.

Registration Control Block (RCB)

Serves as an input control block for the user of the Directory ME's registration services. It is a record with several fields/parameters that serve as input parameters.

Relay

Process occuring when CDCNET receives a data unit from a directly connected network solution and transmits the data unit to another directly connected network solution.

Remotely Connected Network Solution

A network solution is said to be remotely connected to a system if that system is not locally connected to that network solution, but a path to the network solution exists. That system can then transmit data on the path to the network solution over one or more hops.

Request Primitive

Service that allows the adjacent higher layer to activate a particular service.

Response Primitive

Service that allows the adjacent higher layer to reply to an indication service request.

RIDUs

See Routing Information Data Units.

Routing Information Data Unit (RIDU)

The data structure used by the Routing Management Entity.

RTC

Refer to Real Time Clock.

S

SAP

Refer to Service Access Point.

SAP Identifier

A 16-bit number that uniquely identifies a SAP for other software components.

Service

An entity that is external to CDCNET but is registered within CDCNET as being capable of conducting input and output with a terminal or with another service. Services have names. Terminal users connecting to a host are connecting to a service. An example of a service is the interactive facility (IAF) on a host.

Service Access Point (SAP)

An exchange point between the services of two adjacent Control Data Network Architecture (CDNA) layers.

Service Module (SVM)

System Service Module. Software that, along with other interfaces and gateways, connects NOS hosts to CDCNET. SVM provides services to establish and terminate Network Products connections.

Service Primitive

An interface between functions or a group of functions in adjacent layers. Provides the mechanism through which a layer provides its services.

Session

1. A connection between two stations for the purpose of moving information.

2. The time during which such a connection exists.

Session Control Layer

OSI layer 5. This layer establishes and controls system-dependent aspects of communications sessions between specific device interfaces. It bridges the gap between the services provided by the transport layer and the operating system.

Session Layer

Refer to Session Control Layer.

SNA

Refer to Systems Network Architecture.

Source Code

The symbolic language used to create a machine code.

SSR

Refer to Stream Service Routine.

Status

Information about the current state of a network component: Device Interface (DI), the hardware components (boards, ports) of a device interface, lines and network solutions connected to the device interface, and device interface software.

Stream Service Routine (SSR)

Software that implements Control Data's Network Architecture layer 2 (the data link layer). A stream service routine enables communication over a specific type of network solution.

SVM

Refer to Service Module.

System

System Address

The unique address assigned to a device interface in the network. The system address corresponds to the system title, so that commands and data sent by system title are received at the proper device interface address. Refer also to system identifier.

System Command Language (SCL)

The NOS/VE command language on which CDCNET network operations, configuration, and terminal user commands are based.

System Identifier

Unique 48 bit-ASCII character string that identifies a particular device interface on a network. Each device interface is assigned a unique identifier from a pool of numbers allocated to Control Data Corporation by the Xerox Corporation.

Systems Network Architecture

IBM standard defining the layers and layer protocols to be used within an IBM network.

T

T-to-A

Refer to Terminal-to-Application.

Task

Any code set within the Executive that is not an Interrupt Service Routine. Each task has a unique stack, intertask message queue, and priority.

TDI

Refer to Terminal Device Interface.

Terminal

An operator input/output device used for communication on the network.

Terminal Device Interface (TDI)

The standard CDCNET device interface variant that interconnects terminals, workstations, and unit record devices with an Ethernet local area network.

Terminal-to-Application (T-to-A)

A type of network processing that enables the exchange of data between applications programs that reside on host computers and user terminals or workstations. In this case, protocol conversions occur so that transmitted data is understood both at the host and at the terminal or workstation.

Text

A sequence of characters forming part of a transmission that is sent from a source to a receiver, containing the information to be conveyed.

Title

A string of 1 throuogh 256 ASCII characters that identifiy a network service component such as a device interface or a gateway. The Directory Management Entity refers to the component by its title.

Transport Layer

Open Systems Interconnection (OSI) layer 4. Provides end-to-end control of a communication session once the path has been established. It allows processes to exchange data reliably and sequentially, independent of which systems are communicating.

Transport Network

That portion of the NOS network architecture Control Data Network (CDNA) that moves data between systems. (Includes both hardware and software.)

Trap

A software interrupt that causes the Executive to enter a specific program.

Trunk

A logical definition of a line and the communications software that allows the line to carry data between communications controllers. These controllers could be device interfaces or devices for other networks.

W

Wild Card Characters

Characters that can be used in place of other characters as variables. Wild card characters can be used to replace single characters, to replace strings of characters, or to match characters to those specified in a list. Wild card characters can be used when searching for CDCNET file names in the CDCNET directory using the NETFM utility.

X

X.25

The Consultative Committee of International Telephone and Telegraph (CCITT) standard for the interface between data terminal equipment and data communications equipment in an X.25 packet switching network.

Network Address Record B

This appendix describes the network address record which is used in most of the procedures described in this manual. This record is defined in the common deck B3DINAD.

```
TYPE
internet_address = record
  system_addr: system_address,
  sap_id: sap_id_type,
recend;

network_id_type = integer,

system_id_type = record
  upper: 0 .. 0ffff(16),
  lower: integer,
recend,
sap_id_type = 0 .. 0ffff(16),

system_address = record
  network_id: network_id_type,
  system_id: system_id_type,
recend,
```

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

T

U

W

X

Comments (continued from other side)

Please fold on dotted line;
seal edges with tape only.

FOLD

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
First-Class Mail Permit No. 8241 Minneapolis, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CONTROL DATA
Technology & Publications Division
ARH219
4201 N. Lexington Avenue
Arden Hills, MN 55126-6198

Network Management Entities and Layer Interfaces 60462420 A

We value your comments on this manual. While writing it, we made some assumptions about who would use it and how it would be used. Your comments will help us improve this manual. Please take a few minutes to reply.

Who are you?

- ☐ Manager
- ☐ Systems analyst or programmer
- ☐ Applications programmer
- ☐ Operator
- ☐ Other ______________________

How do you use this manual?

- ☐ As an overview
- ☐ To learn the product or system
- ☐ For comprehensive reference
- ☐ For quick look-up

What programming languages do you use? ______________________

How do you like this manual? Check those questions that apply.

Yes	Somewhat	No	
☐	☐	☐	Is the manual easy to read (print size, page layout, and so on)?
☐	☐	☐	Is it easy to understand?
☐	☐	☐	Does it tell you what you need to know about the topic?
☐	☐	☐	Is the order of topics logical?
☐	☐	☐	Are there enough examples?
☐	☐	☐	Are the examples helpful? (☐ Too simple? ☐ Too complex?)
☐	☐	☐	Is the technical information accurate?
☐	☐	☐	Can you easily find what you want?
☐	☐	☐	Do the illustrations help you?

Comments? If applicable, note page and paragraph. Use other side if needed.

Would you like a reply? ☐ Yes ☐ No

From:

Name	Company
Address	Date
	Phone

Please send program listing and output if applicable to your comment.

CONTROL DATA